MASSACHUSETTS INSTITUTE OF TECHNOLOGY
ARTIFICIAL INTELLIGENCE LABORATORY

A.I. Memo No. 606 December, 1980

Automatic Planning of Manipulator Transfer Movements

Tomás Lozano-Pérez

ABSTRACT. This paper deals with the class of problems that involve finding where to place or how to move a solid object in the presence of obstacles. The solution to this class of problems is essential to the automatic planning of manipulator transfer movements, i.e. the motions to grasp a part and place it at some destination. This paper presents algorithms for planning manipulator paths that avoid collisions with objects in the workspace and for choosing safe grasp points on objects. These algorithms allow planning transfer movements for cartesian manipulators. The approach is based on a method of computing an explicit representation of the manipulator configurations that would bring about a collision [27].

Acknowledgements. This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the laboratory's artificial intelligence research is provided in part by the Office of Naval Research under Office of Naval Research contract N00014-77C-0389.

1. Introduction

An important goal of research on programming languages for computer-controlled manipulators is a language in which assembly operations can be concisely described. Two major approaches to manipulator programming have been identified [34]:

1. Explicit programming — in which the user specifies all the manipulator motions needed to accomplish a task.
2. Model-based programming — in which the user specifies geometric models of parts and a description of the task in terms of these models. The detailed manipulator motions are derived by the assembly system from these specifications.

This paper presents algorithms for some of the central geometric problems that arise in the model-based approach to manipulator programming. In particular, this paper deals with the class of problems that involve finding where to place or how to move a solid object in the presence of obstacles. The solution to this class of problems is essential to the automatic planning of manipulator transfer movements, i.e. the motions to grasp a part and place it at some destination. For example, planning transfer movements requires the ability to plan paths for the manipulator that avoid collisions with objects in the workspace and the ability to choose safe grasp points on objects. The approach to these problems described here is based on a method of computing an explicit representation of the manipulator configurations that would bring about a collision [27].

Several model-based manipulator systems have been described in the recent literature: AL [10] [46], Autopass [24] [28] [50], LAMA [25] [26] and RAPT [38] [39] [40]. These are experimental systems, currently under development[1] . Work on the model-based aspects of AL has focused on techniques for making coding decisions in manipulator programs. The decisions are made among a fixed set of strategies so as to minimize estimated execution times and so as to bring estimates on the accuracy of part positions within specified bounds. A central technical issue in this approach is deriving the accuracy estimates from geometric relationships and local accuracy information. RAPT has focused on the specification of manipulator programs by specifying the desired symbolic spatial relationships among objects. These relations are then translated into algebraic constraints on the position

[1]The AL language, as originally described, includes explicit as well as model-based programming capabilities. The former are currently available, while the latter are still in the experimental stage.

parameters of the objects, which can be solved by symbolic manipulation. These algebraic solution techniques are also used to complete the specification of partially specified actions so as to achieve the desired relationships. Implementation work on LAMA and Autopass has focused on techniques for planning collision-free motions, e.g. grasping and parts transfer motions, using polyhedral object models. The techniques reported in this paper are extensions of the Autopass obstacle avoidance algorithm and LAMA's grasping strategies.

A number of important problems relevant to model-based manipulator programming have been addressed independently of any manipulator system, for example the problem of specifying compliant motion strategies based on geometric and kinematic models of a task [30], the selection of grasping positions [5] [31] [35] [51], and the problem of collision detection and collision avoidance among obstacles [3] [7] [12] [33] [47].

The algorithms discussed in this paper are based on previous work on obstacle avoidance algorithms. In particular, [48] [49] first formulated the obstacle avoidance problem in terms of an obstacle transformation which allows treating the moving object as a point. A similar transformation was also used in [1] [2] [4] [45] for the template layout problem; related applications are also discussed in [11] [16]. Generalizations of these obstacle transformation techniques and a review of related work can be found in [27] and [28]. Other approaches to automatic obstacle avoidance are reviewed in [23] [48].

2. The "Pick and Place" Synthesis Problem

The most common transfer movements are of the "pick and place" type, consisting of (1) moving the manipulator from its current configuration[2] to a grasp configuration on some object, P, (2) grasping P, and (3) moving P to some specified configuration. The "pick and place" synthesis problem is that of deriving the manipulator motions that will carry out a "pick and place" transfer movement, given as input the following data:

1. a geometric description of the manipulator and the objects in the workspace,
2. the current configurations of the manipulator and the objects in the workspace,

[2] *Configuration* will be used here to refer to the combined position and orientation of an object as well as to the set of joint parameters specifying the arrangement of manipulator links.

3. the desired final configuration of P, and

4. (Optional) the grasp configuration on P.

This paper focuses on the geometric aspects of the "pick and place" synthesis problem. For example, when the grasp configuration is known, the "pick and place" synthesis problem is equivalent to finding collision-free paths for the manipulator and P between the configurations in items 2, 3 and 4 above; when the grasp configuration is unknown, there is the additional task of choosing a configuration such that:

1. the manipulator's fingers are in contact with P,

2. the manipulator does not collide with nearby objects,

3. the configuration is reachable, and

4. the object is stable in the manipulator's hand.

The first three conditions reflect geometric constraints on the manipulator configuration, relative to P and to other objects in the workspace. The stability condition reflects aspects of grasping beyond the purely geometric, but when P is small relative to the manipulator hand and when parts mating effects are ignored, then stability considerations can typically be reduced to geometric heuristics (see Section 9.6).

The geometric aspects of "pick and place" can be formulated in terms of two fundamental *spatial planning* problems [27], Findspace and Findpath, which occur in many applications. The definition of these basic problems are presented below for the case of polyhedral objects.

Let R be a convex polyhedron that bounds the workspace and which contains k_B other, possibly overlapping, convex polyhedra B_j designated as *obstacles*. Let A, the object being moved, be the union of k_A convex polyhedra A_i, i.e. $A = \bigcup_{i=1}^{k_A} A_i$.

1. **Findspace** — Find a configuration for A, inside R, such that $\forall i \forall j : A_i \cap B_j = \emptyset$. This is called a *safe* configuration.

2. **Findpath** — Find a path for A from configuration s to configuration g such that A is always in R and A never overlaps any of the B_j. This is called a *safe* path.

Clearly, "pick and place" with known grasp configuration can be viewed as a sequence of two Findpath problems. In addition, the configurations which are legal candidates for grasping can be derived from solutions to the Findspace problem.

The reduction of the "pick and place" problem to these more fundamental geometric problems assumes that the locations of all objects are known to high accuracy and that the path of the manipulator can be controlled to the same precision. In a realistic environment, there is always uncertainty in the positions of objects and error in the control of the manipulator. Section 10 discusses the effects of uncertainty.

3. The *Cspace* Approach to Spatial Planning: Overview

In this section, an overview of the Configuration Space approach to spatial planning will be presented; further details can be found in [27].

The position and orientation of a rigid solid can be specified by a single 6-dimensional vector, called its *configuration*. The 6-dimensional space of configurations for a solid, A, is called its Configuration Space and denoted $Cspace_A$. For example, a configuration may have one coordinate value for each of the x, y, z coordinates of a selected point on the object and one coordinate value for each of the object's Euler angles [21]. In general, an n-dimensional configuration space can be used to model any system for which the position of every point on the object(s) can be specified with n parameters. An example is the configuration of an industrial robot with n joints, where n is typically 5 or 6. In $Cspace_A$, the set of configurations of A where A *overlaps* B, i.e. $A \cap B \neq \emptyset$, will be denoted $CO_A(B)$, the $Cspace_A$ *Obstacle* due to B. Similarly, those configurations of A where A is completely *inside* B, i.e. $A \subseteq B$, will be denoted $CI_A(B)$, the $Cspace_A$ *Interior* of B. Together, these two $Cspace_A$ constructs embody all the information needed to solve Findspace and Findpath problems. Note that $CI_A(B) = -CO_A(-B)$, where $-X$ denotes the set complement of X in R.

3.1. Fixed Orientation of A

In two dimensions, if the orientation of a convex polygon A is fixed, $Cspace_A$ is simply the (x, y) plane. This is so because the (x, y) position of some *reference vertex* rv_A is sufficient to specify the

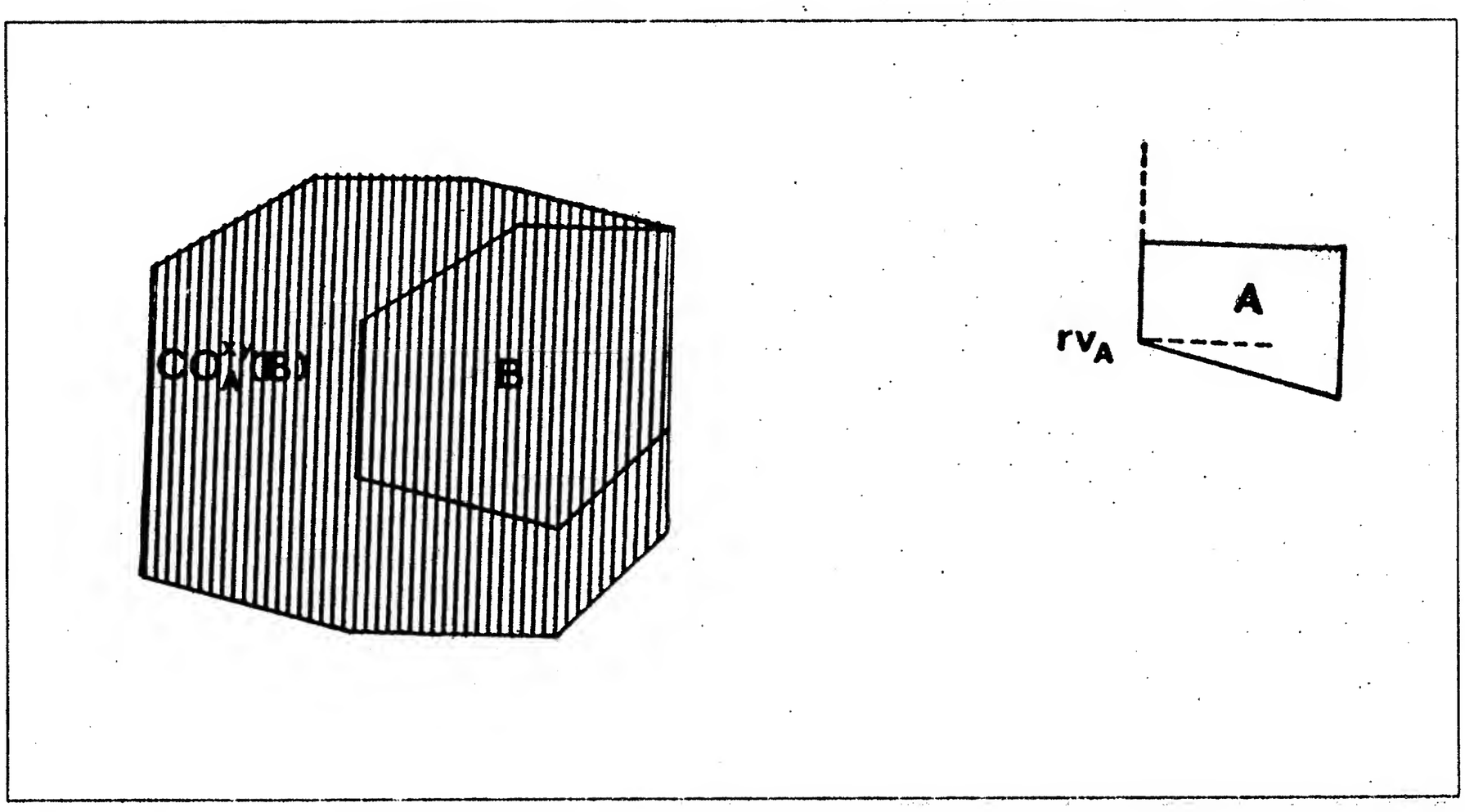


Figure 1. The $Cspace_A$ obstacle due to B, for fixed orientation of A.

polygon's configuration. In this case, the presence of another convex polygon B constrains rv_A to be outside of $CO_A(B)$, a larger convex polygon, shown as the shaded region in Figure 1. Since $CO_A(B)$, in this case, is a set of (x, y) values, it is denoted $CO_A^{xy}(B)$. Similarly, if A and B are three-dimensional polyhedra in fixed orientations, then the *Cspace* obstacles are denoted $CO_A^{xyz}(B)$. Thus, the Findspace problem for polygons, with fixed orientation, can be transformed to the equivalent problem of placing rv_A outside of $CO_A^{xy}(B)$, but inside $CI_A^{xy}(R)$. Similarly, for multiple obstacles B_j, a location for A is safe if and only if rv_A is not inside any of the $CO_A^{xy}(B_j)$, but is inside $CI_A^{xy}(R)$.

If the orientation of A is fixed, then the Findpath problem for the polygon A among the B_j is equivalent to the Findpath problem for the point rv_A among the $CO_A^{xy}(B_j)$. When the $CO_A^{xy}(B_j)$ are polygons, the shortest[3] safe paths for rv_A are piecewise linear paths connecting the start and the goal via the vertices of the $CO_A^{xy}(B)$ polygons, Figure 2. Therefore, Findpath can be formulated as a graph search problem. The graph is formed by connecting all pairs of vertices of $Cspace_A$ obstacles (and the start and goal) that can "see" each other, i.e. can be connected by a straight line that does not intersect any of the obstacles. The shortest path from the start to the goal in this *visibility graph*

[3]This assumes Euclidean distance as a metric. For the optimality conditions using a rectilinear (Manhattan) metric, see [22].

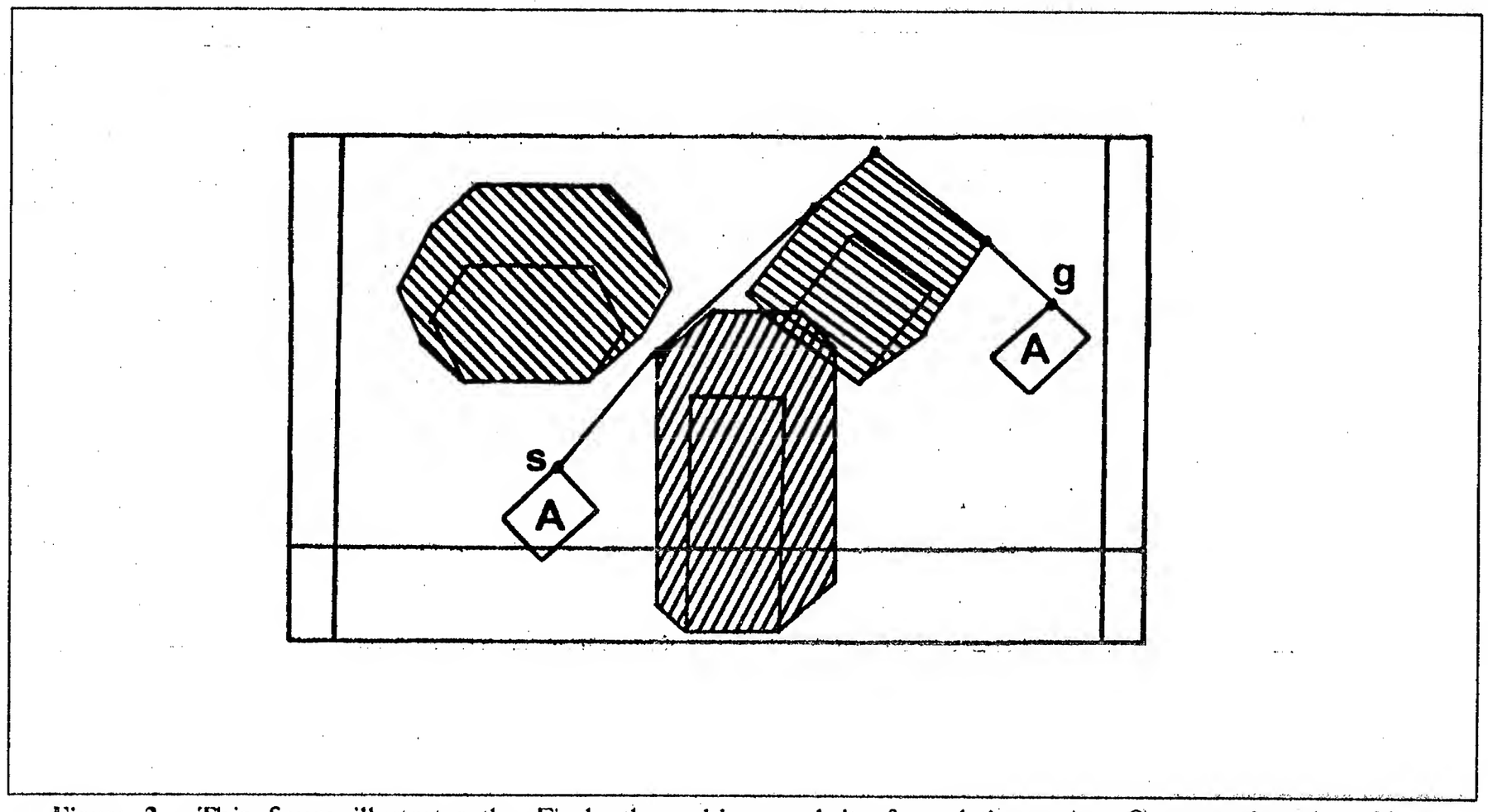


Figure 2. This figure illustrates the Findpath problem and its formulation using *Cspace* obstacles. Note that the shortest collision-free paths connect the origin and the destination via the vertices of the $Cspace_A$ obstacles.

(Vgraph) is the shortest safe path for A among the B_j [28]. This algorithm solves Findpath problems when the orientation of A is fixed. But, because they require moving A along obstacle boundaries, shortest paths are very susceptible to inaccuracies in the object models.

The approach to Findspace and Findpath described above generalizes to problems involving three dimensional polyhedra with fixed orientation. The generalization requires the use of a three-dimensional $Cspace_A$, representing the space of (x, y, z) positions of rv_A. In this *Cspace*, the obstacles are also polyhedra, denoted $CO_A^{xyz}(B)$. However, the Vgraph algorithm has several additional drawbacks when the obstacles are three-dimensional:

1. Shortest paths do not typically traverse the vertices of the $CO_A^{xyz}(B_j)$.

2. There may be *no* paths via vertices, within the enclosing polyhedral region R, although other types of safe paths within R may exist.

These drawbacks may be alleviated by introducing additional nodes in the Vgraph which do not correspond to vertices [28]. An alternative strategy for finding safe paths among two- or three-dimensional $Cspace_A$ obstacles is discussed in Section 7.

3.2. *Algorithms for* $CO_A^{xyz}(B)$

The central operation in the *Cspace* approach to Findspace and Findpath in two and three dimensions is computing $CO_A^{xy}(B)$ and $CO_A^{xyz}(B)$ respectively. If A and B are convex polyhedra, it is simple to show [27] that

$$CO_A^{xyz}(B) = B \ominus (A)_0 = conv\big(vert(B) \ominus vert((A)_0)\big)$$

where $conv(X)$ is the *convex hull* of X [14], $vert(X)$ is the set of vertices of the polyhedron X, $X \ominus Y = \{x - y \mid x \in X \text{ and } y \in Y\}$ and $(X)_0$ means the polyhedron X in its initial configurations, where rv_X is at the origin. This result and the existence of $O(n \log n)$ convex hull algorithms for finite sets of points in $\Re^3$ [41], lead directly to an $O(v^2 \log v)$ algorithm for $CO_A^{xyz}(B)$, where $v = |vert(A)| + |vert(B)|$. The result also holds when A and B are convex polygons, but more efficient algorithms exist for this case. In particular, an $O(v)$ algorithm for $CO_A^{xy}(B)$ is described in [27].

3.3. *Variable Orientation of A*

When A is a three-dimensional solid which is allowed to rotate, $CO_A(B)$ is a complicated curved object in a six-dimensional $Cspace_A$. Rather than compute these objects directly, the approach taken here is to use a sequence of two- and three-dimensional objects to approximate the high-dimensional $Cspace_A$ obstacles. In particular, a six-dimensional $Cspace_A$ obstacle for a rigid solid can be approximated by projections of its (x, y, z)-slices. A j-slice of an object $C \in \Re^n$ is defined to be:

$$\{(\beta_1, \ldots, \beta_n) \in C \mid \gamma_j \leq \beta_j \leq \gamma'_j\}$$

Where γ_j and γ'_j are the lower and upper bounds of the slice, respectively. Then, if K is a set of indices between 1 and n, a K-slice is the intersection of all the j-slices for $j \in K$. Notice that a K-slice of C is an object of the same dimension as C. Slices can then be *projected* onto those coordinates not in K to obtain objects of lower dimension.

Figure 3 shows a two-dimensional example of slice projection. The objects shown shaded represent the (x, y) projection of three θ-slices of $CO_A(B)$ when A and B are convex polygons. These slices

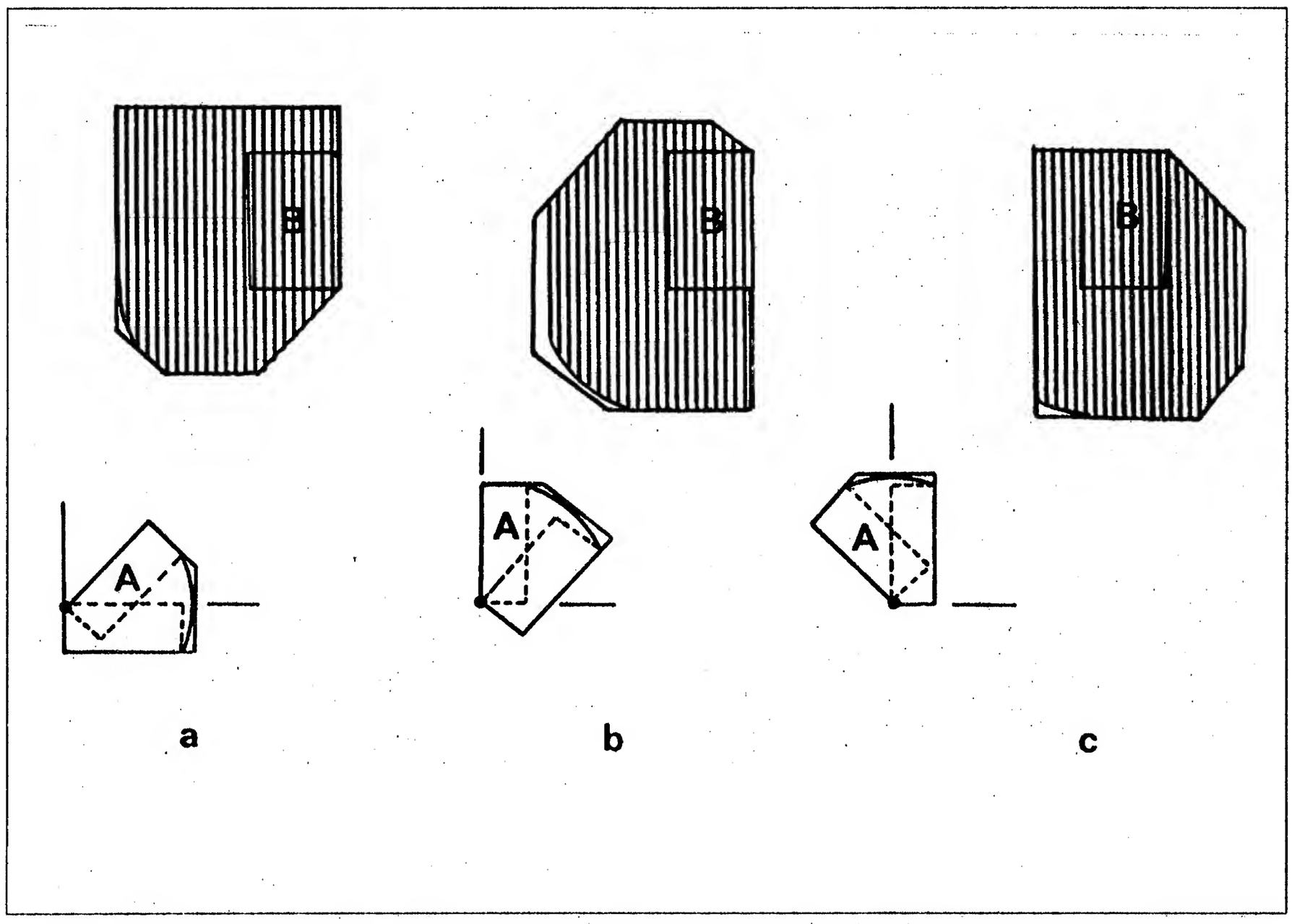


Figure 3. Slice projections of $Cspace_A$ obstacles computed using the (x, y)-area swept out by A over a range of θ values. Each of the shaded obstacles is the (x, y)-projection of a θ-slice of $CO_A(B)$. The figure also shows a polygonal approximation to the slice projection and the polygonal approximation to the swept volume from which it derives.

represent configurations where A overlaps B for some orientation of A in the specified range of θ. In [27] is a proof that these slice projections are equivalent to the CO^{xy} of the area (volume) swept out by A over the range of orientations of the slice. Note that approximating the swept volume as a polyhedron leads to a polyhedral approximation for the projected slices of the $Cspace_A$ obstacles, as shown in Figure 3.

Slice projection has two important properties:

1. A solution to a Findspace problem in any of the slices is a solution to the original problem, but since the slices are an approximation to the $Cspace_A$ obstacle, the converse is not necessarily true.

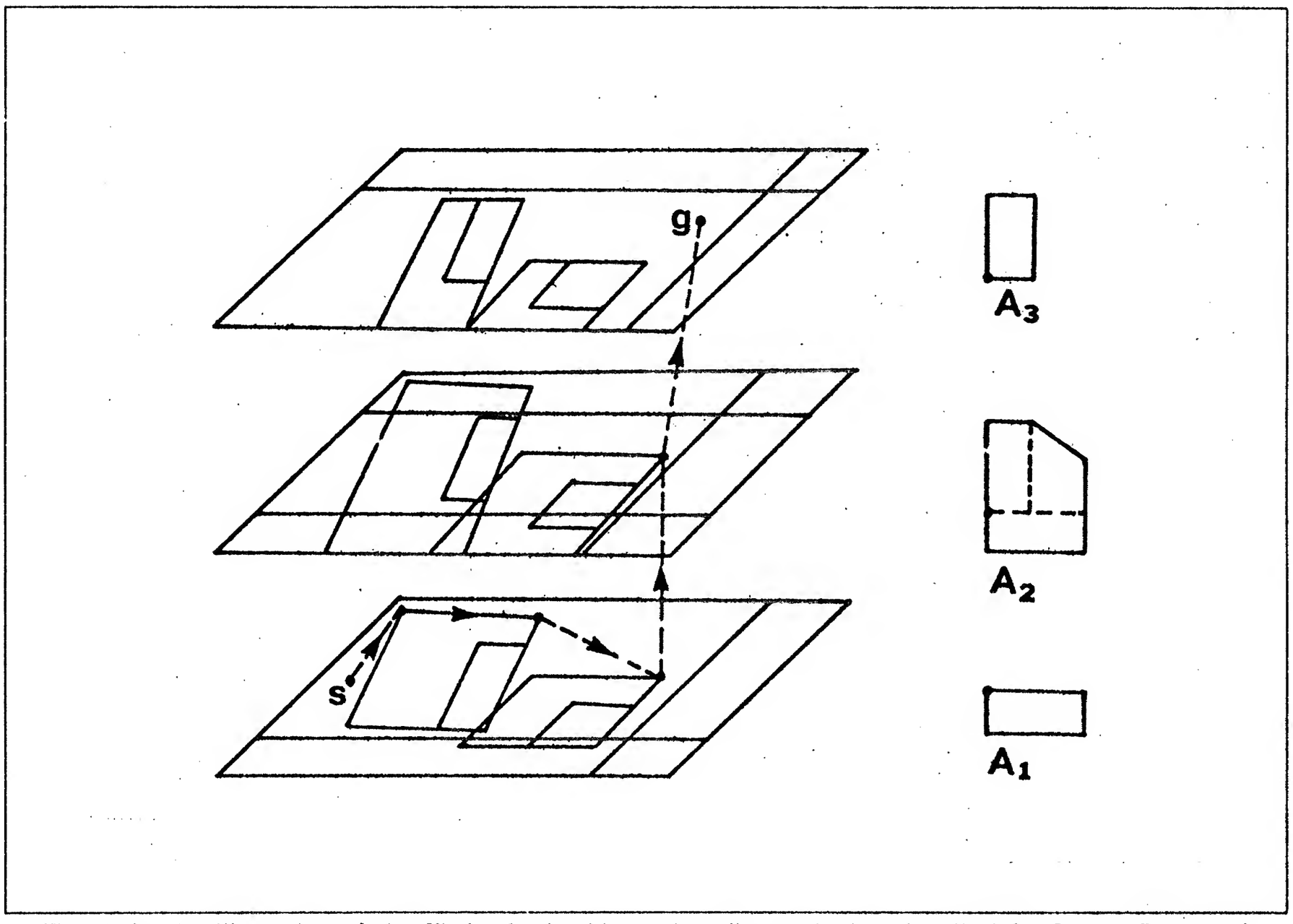


Figure 4. An illustration of the Findpath algorithm using slice projection described by Lozano-Perez and Wesley in [28]. A number of slice projections of the *Cspace* obstacles are constructed for different ranges of orientations of A. The problem of planning safe paths in the high-dimensional $Cspace_A$ is decomposed into (1) planning safe paths via CO vertices within each slice projection and (2) moving between slices, at configurations that are safe in both slices. A_1 represents A in its initial configuration, A_3 represents A in its final configuration and A_2 is a simple polyhedral approximation to the swept volume of A between its initial and final orientation.

2. The slice projection of a $Cspace_A$ obstacle can be computed, by using the swept volume operation, without having to compute the high-dimensional $Cspace_A$ obstacle, see Section 5.

When rotations of A are allowed, the slice projection operation can be used to extend the Vgraph algorithm described earlier to find safe (but sub-optimal) paths [28]. A number of slice projections of the $Cspace_A$ obstacles are constructed for different ranges of orientations of A. The problem of planning safe paths in the high-dimensional $Cspace_A$ is decomposed into:

1. planning safe paths via the vertices of $Cspace_A$ obstacles within each slice projection, and
2. moving between slices, at configurations that are safe in both slices.

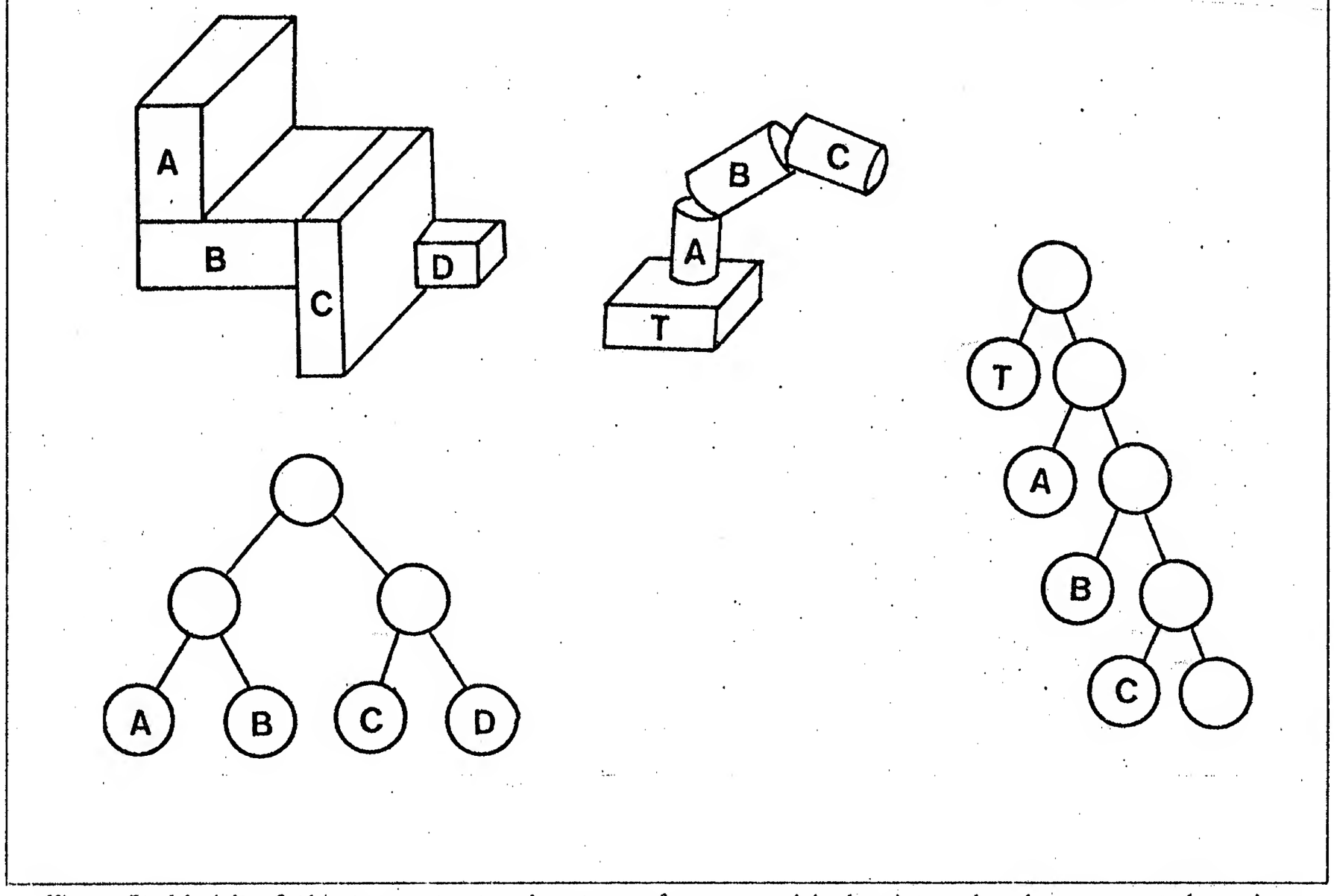


Figure 5. Models of objects are structured as trees of convex polyhedra: internal nodes represent the union of their sub-trees. Linked polyhedra are used to represent manipulators; internal nodes represent joints and the leaves represent links. The nesting of sub-trees in the models of linked polyhedra reflect the cascading effect of joint motions.

Both of these types of motions can be modelled as links in the Vgraph, therefore the complete algorithm can be formulated as a graph search problem. This approach is illustrated in Figure 4. However, since the obstacles are three-dimensional, the Vgraph algorithm is subject to the drawbacks described earlier.

4. Findpath for Cartesian Manipulators

This section overviews an implementation[4] of the Findpath algorithm, for cartesian manipulators (see definition below). Sections 5 through 8 present a more detailed description of the implementation. The system inputs are:

[4]The current implementation is written in LISP for the MIT LISP Machines.

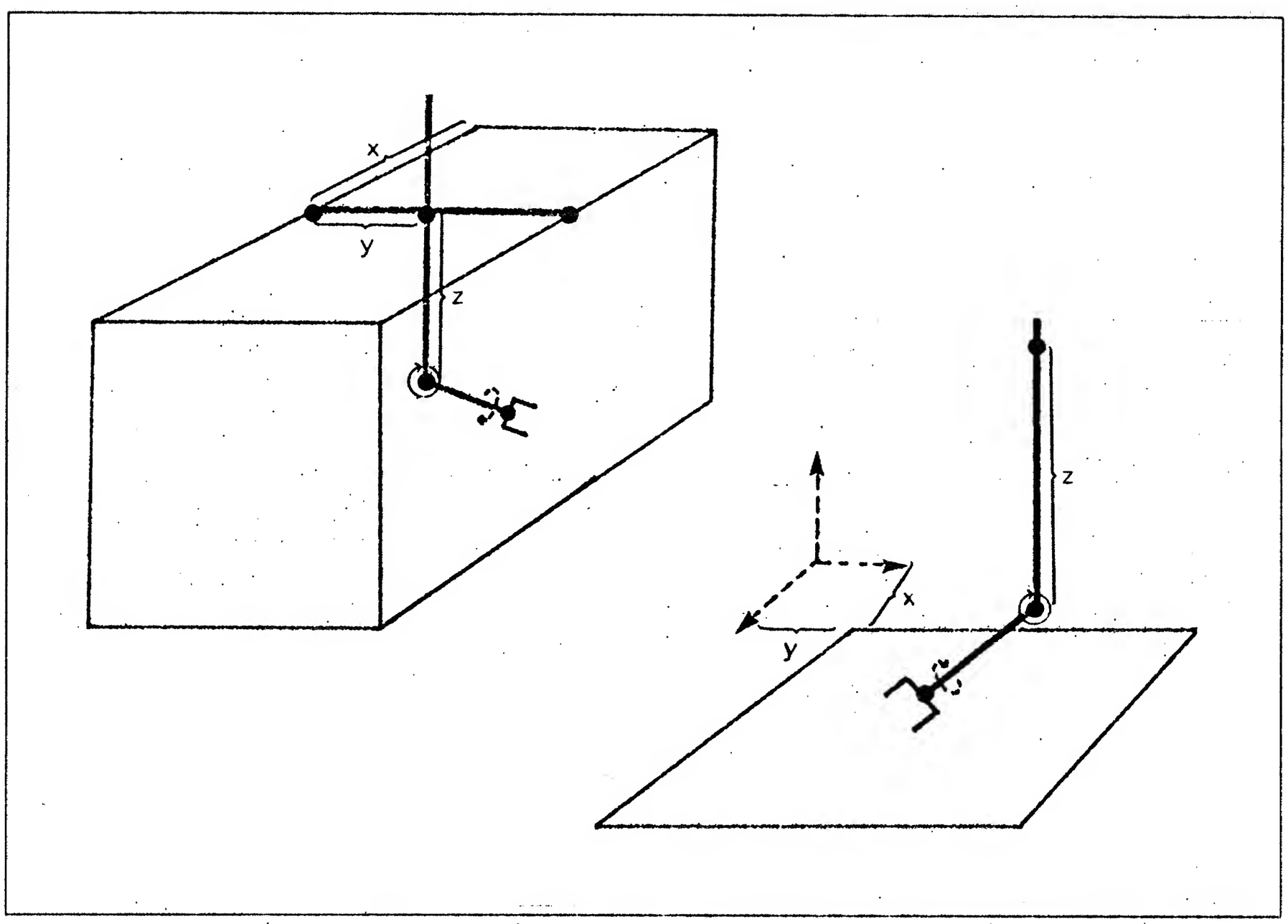


Figure 6. A schematic representation of the link arrangement in two types of existing cartesian manipulators.

1. A polyhedral model of the workspace — where each object is represented by a tree of convex polyhedra, see Figure 5(a).

2. A polyhedral model of the manipulator — represented as a set of link bodies connected by rotary or prismatic joints, see Figure 5(b).

3. A kinematic model of the manipulator — currently, partly embedded in procedures which apply to the polyhedral model and partly in the model structure.

4. A start and a goal configuration for the manipulator.

The system output is a safe path from the start to the goal configurations of the manipulator. The paths are composed of a sequence of linear segments in the *Cspace* of the manipulator.

The implementation described here is limited to cartesian manipulators, i.e. those having three

perpendicular translational degrees of freedom corresponding to the x, y and z axes and up to three rotary degrees of freedom, usually centered at the wrist. Figure 6 illustrates two different types of cartesian manipulators. The restriction to cartesian manipulators allows the use of the $CO_A^{xyz}(B)$ algorithm described in Section 3.2 as the main tool for capturing path constraints.

The Findpath algorithm carries out the following steps in turn:

1. Constructing the $Cspace_A$ obstacles — The slice projections of the $Cspace_A$ obstacles approximate the constraints on the configurations of the manipulator due to the presence of objects in the manipulator's workspace, see Section 5.

2. Representing free space — Once the $Cspace_A$ obstacles are known, the system computes a decomposition of the space outside these obstacles into convex polyhedral cells; these polyhedra are then linked into a graph structure called the Free Space Graph. Each node of the graph represents a free space cell and a link between cells indicates that they touch or overlap, see Section 6.

3. Searching for a safe path — The Free Space Graph is searched to locate a *cell path*, a connected set of free space cells that join the origin and the destination. From the cell path, the system derives a *line path*, a piecewise linear path in the manipulator's $Cspace$, see Section 7.

5. Computing the $Cspace_A$ Obstacles

The first and most important step in the Findpath algorithm is that of computing the $Cspace_A$ obstacles arising from the presence of objects in the workspace. The $Cspace_A$ currently used by the system is the seven dimensional joint space of the manipulator, i.e. x, y, and z displacements, the three wrist rotations and the finger opening. The $Cspace_A$ obstacles are complicated objects in this high-dimension space. To avoid having to deal directly with these objects, the system makes use of slice projection to approximate the $Cspace_A$ obstacles by a set of three-dimensional obstacles.

The $CO_A^{xyz}(B)$ algorithm of Section 3.2 computes an (x, y, z) cross-section of $CO_A(B)$ for a specified orientation of A. But, this algorithm can be adapted to compute the (x, y, z)-slice projections of $CO_A(B)$. The construct that relates slice projections to the cross-sections is the *swept volume* of an object. The swept volume of A is the union of $(A)_a$, i.e. A in configuration a, for a within the

configuration range denoted by $[c, c']_K$, where c and c' are configurations of A and K is a subset of the configuration parameters. A configuration a is in the range $[c, c']_K$ if, for each i in K, the i^{th} parameter of a is between the i^{th} parameters of c and c'. For example, if c and c' are of the form $(\beta_1, \beta_2, \beta_3)$ and $K = \{3\}$, then the swept volume of A over the range $[c, c']_K$ refers to the union of A over a set of configurations differing only on β_3. The swept volume of A over this configuration range is denoted $A[c, c']_K$. It can be shown [27] that the (x, y, z)-slice projection of $CO_A(B)$ over the orientation range contained in $[c, c']_K$ is the same as $CO^{xyz}_{A[c,c']_K}(B)$.

In summary, the computational requirements of the slice projection technique are:

1. Choosing a decomposition of the orientation ranges of the cartesian manipulator into sub-ranges, $[c, c']_K$, to be used for slice projection.

2. Computing polyhedral approximations to $A[c, c']_K$ for each orientation range.

3. Computing $CO^{xyz}_{A[c,c']_K}(B_j)$ for each obstacle B_j and each orientation range.

This section addresses these issues. First we assume that the orientation ranges defining the slices are given; the discussion of choosing slice parameters will be taken up at the end of the section.

5.1. Computing the Swept Volume for Linked Polyhedra

The swept volume of a polyhedron A over a range of translations is another polyhedron. Let $T \subseteq I$ be the set of configuration parameters corresponding to the translations of A. If A is a convex polyhedron and the range of positions of the reference vertex of A over the range of translations $[c, c']_K$ can be represented as a convex polyhedron V, then $A[c, c']_T = A \oplus V$ where $X \oplus Y = \{ x + y \mid x \in X \text{ and } y \in Y \}$. Since $A \oplus V = conv(vert(A) \oplus vert(V))$, this leads to a direct algorithm for computing the swept volume for translation. If the range of configurations includes rotations then the swept volume is not a polyhedron. In the rest of the paper it is assumed that a polyhedral approximation to the swept volume is always available. The Appendix describes an algorithm to compute a simple approximation to the swept volume of a convex polyhedron under pure rotation.

The swept volume of A, a rigid object, resembles another rigid object with the same number of degrees of freedom. But for manipulators, modelled as linked polyhedra, the situation is more

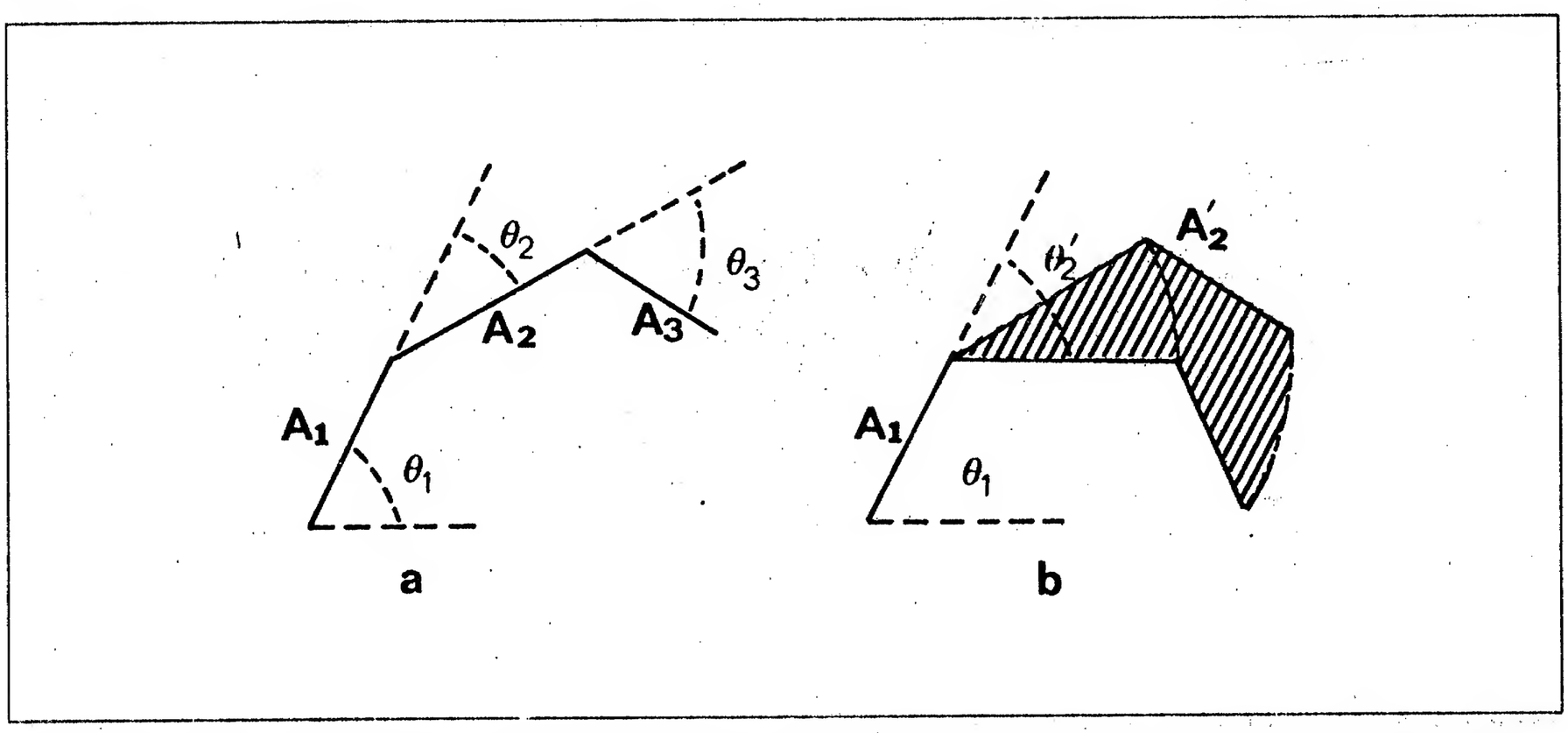


Figure 7. Changes in the second joint angle from θ_2 to θ'_2 causes changes in the configurations of both link A_2 and link A_3.

complex. Linked polyhedra are kinematic chains with polyhedral links and prismatic or rotary joints. The relative position and orientation of adjacent links, A_i and A_{i+1}, is determined by the i^{th} *joint parameter (angle)* [36]. The set of joint parameters of a linked polyhedron completely specifies the position and orientation of all the links.

Note that for a linked polyhedron, the position of link j typically depends on the positions of links $k < j$, which are closer to the base than link j. Let $K = \{j\}$, $c = (\theta_i)$, $c' = (\theta'_i)$, and $[c, c']_K$ define a range of configurations differing on the j^{th} $Cspace_A$ parameter. Since joint j varies over a range of values, links $l \geq j$ will move over a range of positions which depend on the values of c and c', as shown in Figure 7. The union of each of the link volumes over its specified range of positions is the swept volume of the linked polyhedron. The swept volume of links j through n can be taken as defining a new j^{th} link. The first $j - 1$ links and the new j^{th} link define a new manipulator whose configuration can be described by the first $j - 1$ joint parameters. On the other hand, the shape of the new link j depends not only on the K-parameters of c and c', i.e. θ_j and θ'_j, but also on θ_l for $l > j$. This implicit dependence on parameters of c and c' that are not in K is undesirable, since it means that the *shape* of the new j^{th} link will vary. Letting $K = \{j, \ldots, n\}$, then the shape of the swept volume depends only on the K-parameters of c and c', while its configuration is determined by the $(I - K)$-parameters. A swept volume that satisfies this property is called *displaceable.*

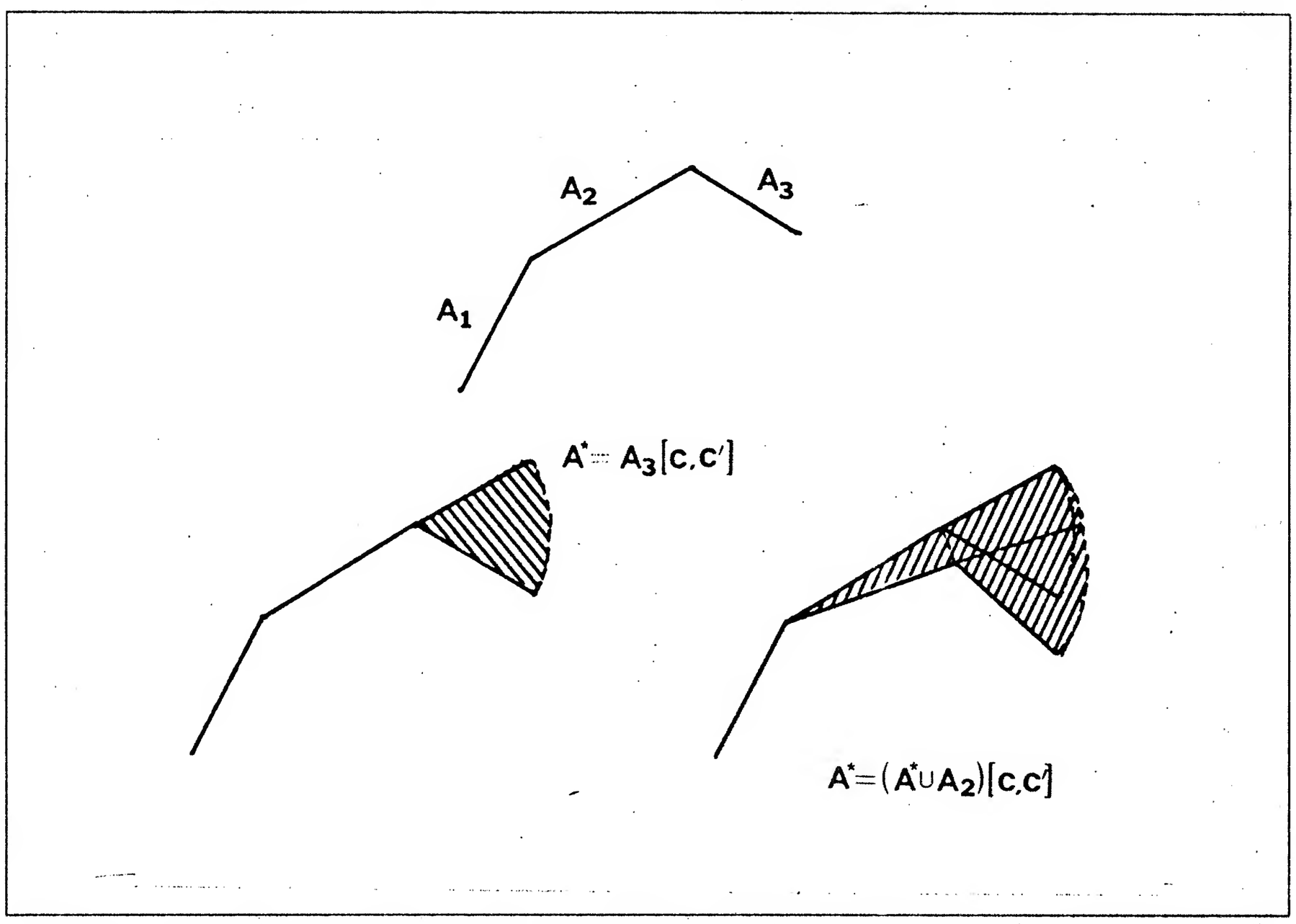


Figure 8. Computing the swept volume for linked polyhedra. If $[c, c']_K$ involves ranges of configurations of the second and third link, first compute the swept volume for the third link and then the swept volume for the union of the second link and the swept volume of the third link.

Given an operation for computing (a polyhedral approximation to) the swept volume of a polyhedron, see Appendix, then this operation is applied to computing the swept volume of linked polyhedra. The swept volume $A[c, c']_K$ is computed by the following process, illustrated in Figure 8:

1. Let $i = n$, where $n > 0$ is the number of links in the linked polyhedra, $|I| = n$; let $A^* = \emptyset$;
2. Place A in configuration c;
3. Let $A^* = A^* \cup A_i$;
4. If $i \in K$ then let $A^* = A^*[c, c']_{\{i\}}$, i.e. update A^* to be the swept volume of A^* over the range of i^{th} joint;
5. Let $i = i - 1$. If $i = 0$ then stop, else go to step 3.

The swept volume obtained in this fashion can then be used to compute the $CO^{xyz}_{A[c,c']_K}(B_j)$.

5.2. *Computing slice projections for* $Cspace_A$ *obstacles*

If $A[c, c']_K$ overlaps some obstacle B then, for some configuration a in the range $[c, c']_K$, $(A)_a$ overlaps B. The converse is also true. If $A[c, c']_K$ is displaceable, then $CO_{A[c,c']_K}(B)$ is the set of $I - K$ projections of those configurations of A within $[c, c']_K$ for which A overlaps B. Equivalently, $CO_{A[c,c']_K}(B)$ is the $I - K$ projection of the $[c, c']_K$ slice of $CO_A(B)$. If A is a cartesian manipulator and K is the index set for the wrist rotations of the manipulator, then the configurations of the swept volume are the (x, y, z) positions of some point on the manipulator. The algorithm of section 3.2 can be used to compute $CO^{xyz}_{A[c,c']_K}(B)$ and thereby compute the required slice projections of $CO_A(B)$.

Given the swept volume of the manipulator model for a particular range of parameters $[c, c']_K$, the next step is to compute the slices of all the *Cspace* obstacles for the manipulator over that range of configurations; this set is denoted $COS[c, c']$. In previous discussions of the $CO^{xyz}_A(B)$ algorithm we have assumed that A and B were single polyhedra; we saw in the previous sections that both the object and manipulator models are structured as part trees, whose leaves are convex polyhedra. The actual model of a manipulator or a part is the union of the fringe, i.e. the set of leaves, of the corresponding part tree. Thus if $A = \bigcup_{i=1}^{k_A} A_i$ and $B = \bigcup_{j=1}^{k_B} B_j$, the following result can be used in computing $CO^{xyz}_A(B)$:

$$CO_A(B) = \bigcup_{i=1}^{k_A} \bigcup_{j=1}^{k_B} CO_{A_i}(B_j),$$

This result means that $k_A \times k_B$ applications of the $CO^{xyz}_{A_i}(B_j)$ algorithm must be carried out to compute $CO^{xyz}_A(B)$ exactly. In the "pick and place" application, an exact model of all the *Cspace* obstacles is not usually needed since the manipulator will not move close enough to all the obstacles.

The amount of time needed to compute the COS can be reduced by simplifying the geometric models of both the A_i and the B_j when appropriate. The current implementation uses a simple family of succesively finer approximations for objects based on the part tree. Consider the part tree for an object B_j, where each of the leaves of the tree is a convex polyhedron. Define a *covering node set* recursively to be either (1) the set containing just the root of the part tree or (2) obtained from

another covering node set by replacing some node, internal to the part tree, with all its descendants. If each internal node represents the union of all its descendants, then every covering node set is a complete model of the object. In practice, internal nodes of the part tree store the bounding rectangular solid[5] of the union of all its descendants instead of the union itself. Thus, the family of covering node sets represents progressively more detailed models of the part [29]. Using these approximations reduces the number of applications of $CO_{A_1}^{xyz}(B_j)$ needed to compute the COS, since the number of polyhedra in a covering node set is less than or equal to that in the full fringe. In addition, it can be used to simplify many of the individual computations, because when A and B are bounding rectangular solids, computing $CO_A^{xyz}(B)$ is trivial. In particular, if the bounding solids are represented by the endpoints of their main diagonal, e.g. $A = (a_1, a_2)$ and $B = (b_1, b_2)$, then $CO_A^{xyz}(B) = (b_1 - (a_2 - a_1), b_2)$.

For simplicity, the current implementation uses a three level part tree for the swept volume of the manipulator and for the objects in the workspace. Each tree has a root node which models the complete object by one bounding rectangular solid. The descendants of the root are bounding rectangular solids for each of the convex components of the model and the leaves of the tree are the convex polyhedra whose union is the complete object model. Therefore if the object is modelled as the union of k convex polyhedra, the part tree has $2k + 1$ nodes. Using this representation, $CO_A^{xyz}(B)$ can be modelled as a tree of similar structure with $2(k_A \times k_B) + 1$ nodes. Any covering node set of this tree is an approximation to the $Cspace_A$ obstacle corresponding to B. In practice, the complete tree is not computed at once, rather the simplest approximation, the bounding rectangular solid of the whole object, is computed and successive covering node sets are computed as needed. This is discussed further in section 6.

5.3. Choosing the Slice Parameters

So far we have assumed that the configuration ranges defining the $Cspace_A$ slices were given as input; in this section, the choice of ranges is discussed. The primary choice is how large to make the ranges, since it is this that affects the system's capability to use changes in the orientation of the hand

[5] A *bounding rectangular solid* for a polyhedron is a rectangular solid whose edges are parallel to the coordinate axes and that completely includes the polyhedron.

to avoid obstacles. In particular:

1. The larger the orientation range of a slice, the larger the manipulator's swept volume, the larger (and less accurate) the $Cspace_A$ obstacles and the fewer the legal configurations and legal motions of the manipulator.

2. The smaller the orientation range of slices, the larger the number of slices needed to cover the $Cspace$ and the more time needed to compute the COS and to search them for a path.

These conflicting effects can be balanced by taking advantage of the fact that, for "pick and place" motions, the accuracy requirements are higher near the start and the goal of the path, where the manipulator is moving near obstacles, than along the rest of the path [28] [48] [49]. This suggests defining slices with small rotation ranges centered around the orientations of the start and the goal; slices with larger ranges may be used for the remaining orientations. This approach is used in the current implementation. In particular, a COS is defined for the orientation of the manipulator in the start configuration and one for the orientation manipulator in the goal configuration; these COS correspond to slices with singular orientation ranges, i.e. where the upper bound of the range equals the lower bound[6]. In addition, the total range of parameters in $Cspace_A$ is divided among some number of other slices[7] each with non-singular ranges. Furthermore, slices with singular ranges are defined for configurations at the intersection of the slice parameters of the "larger" slices. This last type of slice allows moving between safe configurations in the "larger" slices.

Note that the computational burden of adding an extra slice is very low if bounding rectangles are used for objects. This sacrifices some of the potential maneuvering space, but gains a *very* large increase in speed. This is the compromise taken in the current implementation.

Motions within a slice with a singular orientation range are limited to translations, while rotation is legal within a slice with non-singular ranges. Therefore, the classes of motions allowed by the system are those composed of translations interspersed with rotations, but where the rotations happen in increments defined by the slices parameters. This means that this approach may fail to find a safe path in situations where:

---

[6]A slice with a singular range is the same as a *cross-section*.

[7]Currently varying between 8 and 64.

1. all safe paths require rotations combined with translations at a finer resolution than that allowed by the slice ranges, and/or

2. the orientation ranges chosen, although adequate in size, do no match those required in the problem.

These problems can be reduced, at the expense of more computation, by using more slices with smaller ranges. But, there exists problems which require continuous rotation along a path. In practice, most robotics applications do not use the very crowded environments that require very high rotation resolution for the "pick and place" motions. The reason for this is that safe paths in such environments are very hard for humans to specify, are subject to positioning errors of the parts and are difficult for most industrial robots to execute reliably at medium or high speeds.

6. Path Searching and Free Space

Having computed the $Cspace_A$ obstacles, it still remains for the system to find a path among these obstacles. This section briefly touches on alternative strategies for finding safe paths.

One approach to finding paths among obstacles is to search for the shortest path between the start and the goal, without considering other constraints. For example, the Vgraph algorithm described in Section 3 follows this approach. But, the approach has some important drawbacks. Shortest paths in $Cspace_A$ move along the boundaries of the $Cspace_A$ obstacles and are, therefore, very susceptible to model inaccuracy and position error. This problem can be alleviated by adding a uniform "safety margin" around the obstacles, but doing so might disqualify some feasible paths. Furthermore, no efficient algorithms currently exist for finding optimal paths among three-dimensional obstacles. Unlike the situation in two dimensions, there is no finite set of points through which shortest paths are guaranteed to pass. Thus, algorithms would have to be based on iterative numerical methods. For these reasons, only heuristic algorithms for finding safe paths will be considered here. These heuristic algorithms require less execution time and can be extended to consider criteria such as safety margins, but they will not find the shortest path.

Another issue is whether the path search is conducted using primarily a representation of the $Cspace_A$ obstacles themselves, as does the Vgraph algorithm, or of the *free space* outside the

obstacles, as in [48] [49]. Although these representations are equivalent, they lead to different heuristic algorithms. The current implementation uses the free space style of algorithm because it simplifies the formulation of different search heuristics, e.g. the use of variable resolution space representations described below.

The remainder of the section deals with the free space representation technique employed in the Findpath implementation. Section 7 discusses the path search algorithm used on this representation.

6.1. A Free Space Representation

The basic goals for a space representation are accuracy, speed and compactness. In addition, it should facilitate heuristics for the task at hand. The most important heuristic for a space representation is to avoid excess detail (and therefore time spent) on parts of the space which do not affect the operation. Therefore, the space representation should not have to maintain a perfectly detailed model everywhere. Instead, it should have the capability of maintaining a rough model and be able to selectively refine [48] [49] subsections to be as detailed as necessary.

A number of proposals exist for representations of space and objects in space [9] [25] [42]; most of these divide the space into a set of *cells*. The proposals can be partially characterized along the following dimensions:

1. Shape uniformity — are all cells equally shaped?
2. Size uniformity — are all cells the same size?
3. Orientation uniformity — are all cells oriented uniformly?
4. Ordering principle — are the cells ordered into an array, multi-list, tree, or graph.

We will not consider representations which use cells of uniform shape and/or size, since they typically require large numbers of cells to achieve sufficient accuracy[8] . Instead, we use a hybrid cell representation employing two types of cells: (1) rectangular solids aligned with the axes and (2) arbitrary convex polyhedra. The idea is to use the simple rectangular cells away from obstacles where representation economy is important and polyhedral cells where high accuracy, e.g. near an obstacle, is

[8]Udupa [48] [49] employed a free space representation which used rectangular cells of variable size. This approach is adequate for motions that do not closely approach the obstacles.

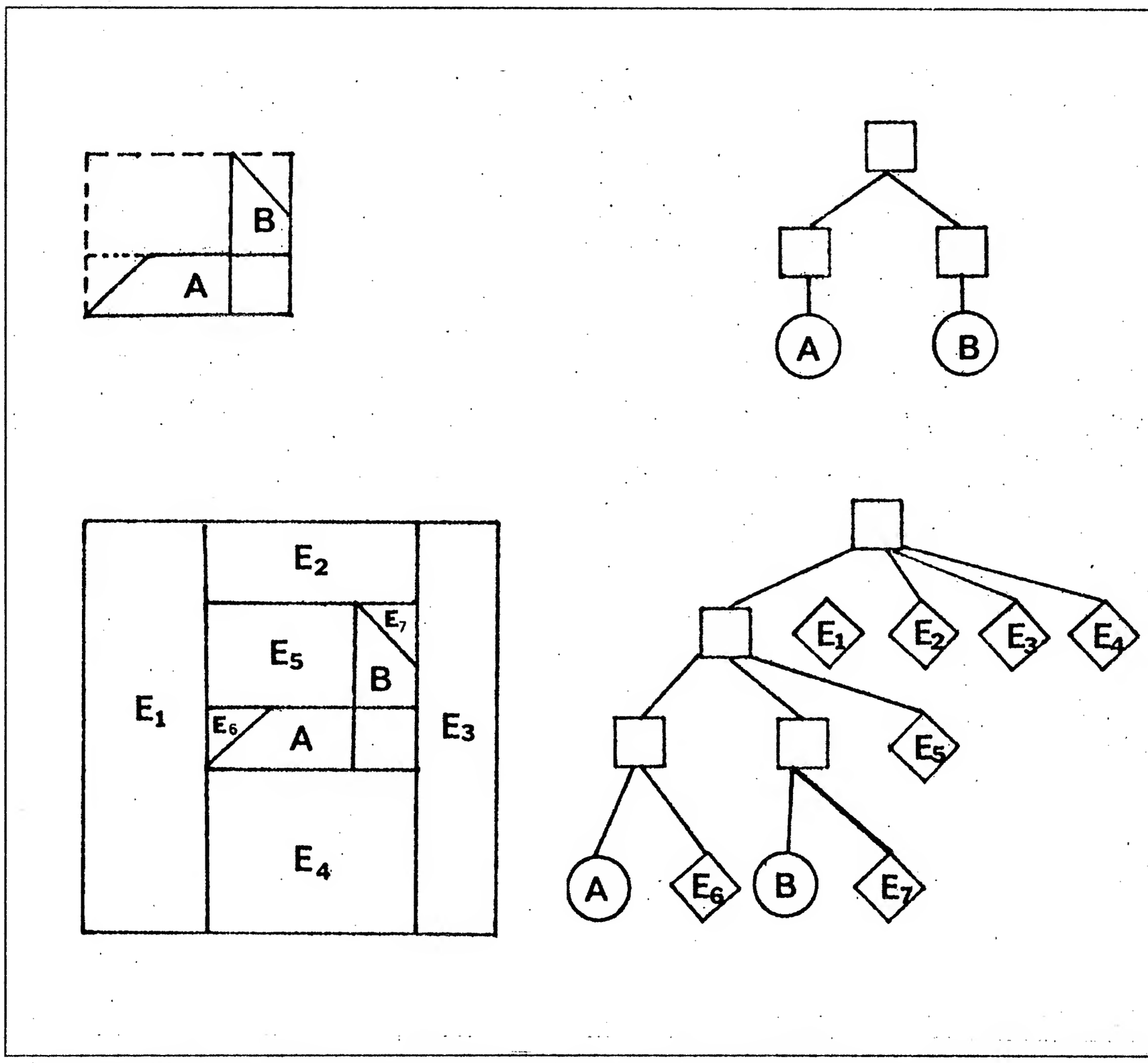


Figure 9. This figure illustrates, in two dimensions, the space representation employed in the implementation of the Findpath algorithm. (a) A sample $Cspace_A$ obstacle with its part representation. (b) The resulting space representation. Rectangular nodes indicate mixed cells, round nodes indicate full cells, and triangular nodes indicate empty cells.

needed.

The space representation described below is analogous to the part representation described earlier, except that a new type of node is introduced. The part tree representation uses rectangular bounding cells as internal nodes and polyhedral cells as leaves. The leaves represent space that is FULL, i.e. completely occupied by an object. The internal cells represent MIXED space, i.e. cells which are part FULL, part EMPTY. But, note that the part tree does not have an explicit representation of the EMPTY space. The space representation simply adds explicit EMPTY cells to the

parts tree representation. Then each internal MIXED node becomes the union of its descendants. In addition, the space representation introduces a new MIXED root node from which all the part representations descend.

The space representation is built up starting with a bounding rectangular solid representing the workspace, this is the first MIXED cell. The descendants of this node are the MIXED cells corresponding to the roots of the trees representing each of the $CO_A^{xyz}(B_j)$, as described in Section 5.2 and a set of EMPTY bounding rectangular solids representing the free space outside the MIXED cells. The representation of each MIXED cell can be further expanded into other EMPTY, MIXED and FULL cells, culminating in a representation involving only EMPTY and FULL convex polyhedral cells as leaves of the tree and MIXED cells as internal nodes, Figure 9. The polyhedral representation of each EMPTY cell must be computed so that it does not overlap any MIXED or FULL cells. As with the part representation, any covering node set of this tree represents a complete model of the space, at some non-uniform resolution. This hybrid cell representation is based on a generalization of the *quad tree* representation used for images [8] [17] [18] [20] [43] and the oct-tree representation of objects [3].

The operations on the space representation described above are very efficient when dealing with bounding rectangular solids. The most expensive operation is when the volume difference of a MIXED rectangular cell and a FULL polyhedral cells must be computed[9] ; this operation results in a description of the EMPTY cells. However, this need only be done when high accuracy is required, usually near the start and the goal of the path. Therefore, the representation meets the criteria stated at the beginning of the section.

6.2. Building a Free Space Graph

The process described in Section 5 produces a slice for each $Cspace_A$ obstacle over each of the orientation ranges, $[c_i, c_i']_K$, of the manipulator's wrist. The set of slices for all obstacles over one orientation range is denoted $COS[c_i, c_i']_K$. For each of these COS_i, a space representation is computed, SR_i, as described above. For each of these SR_i, a Free Space Graph is built, FSG_i, this is a graph where each node is an EMPTY cell in the SR_i and a link indicates that the cells touch or

[9]The current implementation of this operation uses repeated applications of a cutting and capping operation [6].

overlap[10] . In addition, it is necessary to add links to each FSG_i that connect to nodes of other FSG_j whose rotation range overlaps that of FSG_i. That is, for EMPTY cells $C_i \in SR_i$ and $C_j \in SR_j$, if there is some configuration c contained in both cells, then links must be placed between C_i and C_j. This is so because the existence of c guarantees that it is possible to pass from any configuration in C_i to any in C_j and *viceversa* while remaining outside all the obstacles in COS_i and COS_j. The resulting composite FSG is then searched for a path, since each path through the graph corresponds to a class of safe paths in $Cspace_A$ and vice-versa.

7. Path Searching

The Findpath problem is to find a path between two points, the start and the goal, while staying in the free space. In the current implementation, this is carried out by the following steps:

1. Choose the largest EMPTY cell in any of the SR_i enclosing the start configuration. Otherwise, choose some MIXED cell containing the start and expand the representation of this MIXED cell into its constituent EMPTY, MIXED and/or FULL cells. If an EMPTY cell contains the start configuration, stop, else repeat. Note that this computes successively finer models, i.e. successive covering node sets, of the specific area around the start without having to expand the complete model or even any complete part tree. If no EMPTY cell is ever found, the task is impossible since the start configuration causes a collision.

2. Perform step 1 for the goal configuration.

3. Construct a Free Space Graph as described in Section 6.2. At this point, the Free Space Graph is in its final form; the current implementation does not refine the space representation further.

---

[10]The current representation allows EMPTY cells to overlap each other but not MIXED or FULL cells.

4. Search for the shortest path in the Free Space Graph from the cell including the start to that including the goal. The graph search operation can be carried out by any of the standard shortest path algorithms [13]; the current implementation uses the A^* algorithm [15]. These shortest path algorithms require that a weight be assigned to each of the links of the Free Space Graph e.g. indicating the time required to traverse the cells. How this may be done is discussed below. If no path exists, this may be due to the approximations and quantizations used in the solution, see Section 7.3.

5. Choose a line path contained in the cell path. This problem is discussed in Section 7.2.

7.1. Assigning Link Weights for the FSG

The definition of an "optimal" path, or even a "good" path, assumes some choice of performance index. The current implementation uses estimated time of travel along the path as the index. If $Cspace_A$ is the manipulator's joint space, then the time to travel between two configurations can be estimated as the maximum time for any of the joints to travel, at the maximum rated joint velocity, between the joint settings at each configuration. The weights assigned to the links in the FSG should therefore reflect the time needed to travel between two overlapping cells along the optimal path. Of course, no weight assignment can actually do this since it requires knowing the complete optimal path.

A simple alternative is to assign to a link the estimated time of travel between the centroids of the cells that it connects. This weighting function has the advantage of being very easy to compute. For small cells it provides a good approximation of the actual time to traverse the cells, but for larger cells it might overestimate or underestimate the actual time, see Figure 10. The current implementation uses the centroid weighting function, but does not divide the large EMPTY rectangular cells into smaller cells; this will be implemented in the near future.

A more complex weighting function, which would typically produce faster paths, is the following: The weight on the link between cell C and C' is assigned the time to traverse C from p, the point of entry to C, to p', the point of entry into C'. The point p' is the one on $C \cap C'$ that minimizes the distance[11] to the line between p and the goal. The initial C is the cell that contains the start

[11] Actually, the difference in time between the straight line path and one going through this point.

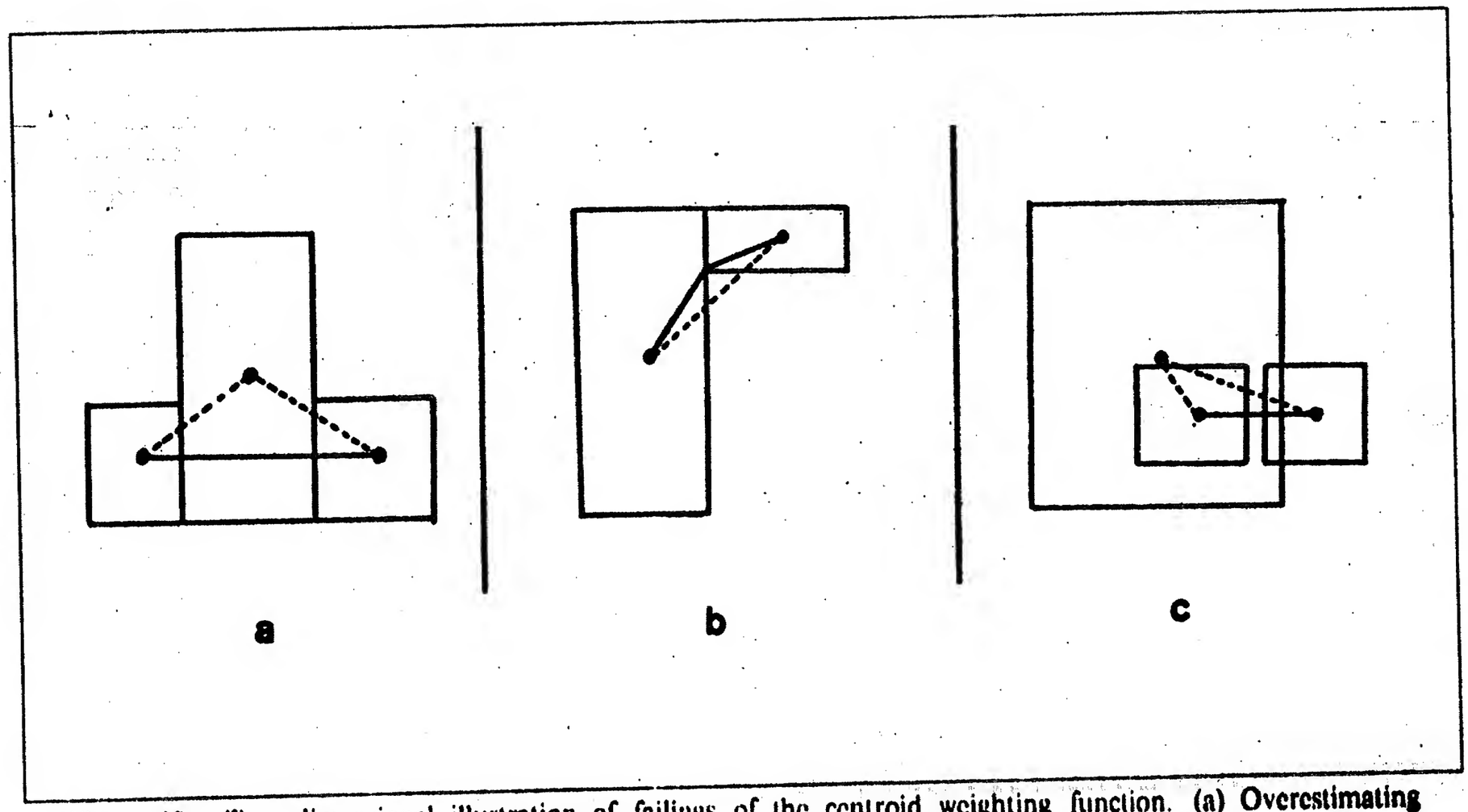


Figure 10. Two dimensional illustration of failings of the centroid weighting function. (a) Overestimating when one cell is large, (b) underestimating because of limited connectivity, and (c) overestimating because of large overlap. The solid line is the optimal path between cells, the dashed lines is the path that the function would use to evaluate the distance between cells.

configuration and the initial p is the start configuration. Clearly, this technique requires much more computation that the centroid weighting described above. For most applications, the simpler centroid function, together with cell splitting should suffice.

7.2. Choosing a Line Path

The search of the FSG produces a list of EMPTY $C space_A$ cells that touch or overlap; it is still necessary to choose a specific path, i.e. some curve, within these cells. The simplest type of path to choose is a piecewise linear one, although the cells simply place configuration constraints on the manipulator along the path and any path satisfying those constraints will be safe.

If the centroid weighting has been used for the links, it is natural to choose a piecewise linear path that traverses the centroids of the cells. Of course, the straight line path between two centroids is not guaranteed to remain within the cells and might therefore not be safe. Therefore an intermediate configuration in the intersection between adjacent cells should be chosen. The centroid of the intersection of adjacent cells on the path can be used for this purpose; this is the technique used in the

current implementation. Alternatively, this point could be chosen so as to minimize the deviation from a straight line path between the centroids. If the cell size is small enough, such paths are adequate for most tasks.

The more complex weighting scheme described earlier produces a sequence of entry points into the cells which may be connected directly to obtain a path. Since the points are contained in the intersection of the cells, a straight line connecting them is guaranted to be in the cell.

7.3. Dealing with Path Search Failure

If the path search algorithm fails to find a safe path, the reason for failure could be one of the following:

1. No safe paths exist.
2. No safe paths exist at the quantization of orientations chosen.
3. The approximations of objects by bounding rectangular solids has removed necessary maneuvering space.

The last two causes of failure may be overcome by decreasing the orientation quantization and/or increasing the representation detail in the space representation, both at the expense of extra computation. This suggests the possibility of increasing the accuracy of the space representation when a path search failure ocurrs. The current implementation does not exploit this possibility.

8. Examples

This section presents output from the implementation running on a simple example. The results are collected in Figure 11.

a. The initial and final configuration of the model, including the manipulator model. Note that the manipulator must rotate to execute this motion.

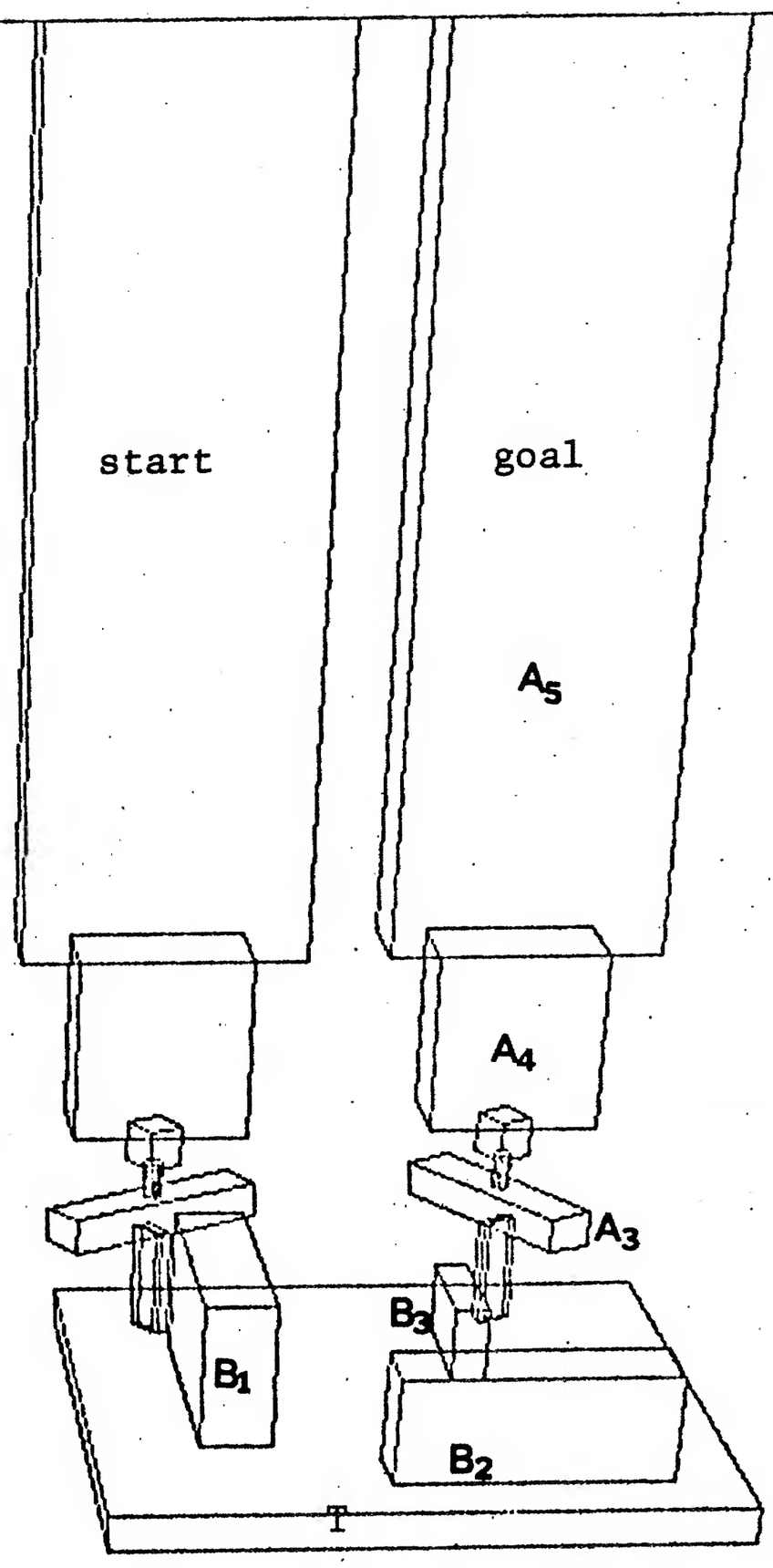


Figure 11a
The start and goal configurations and the world model

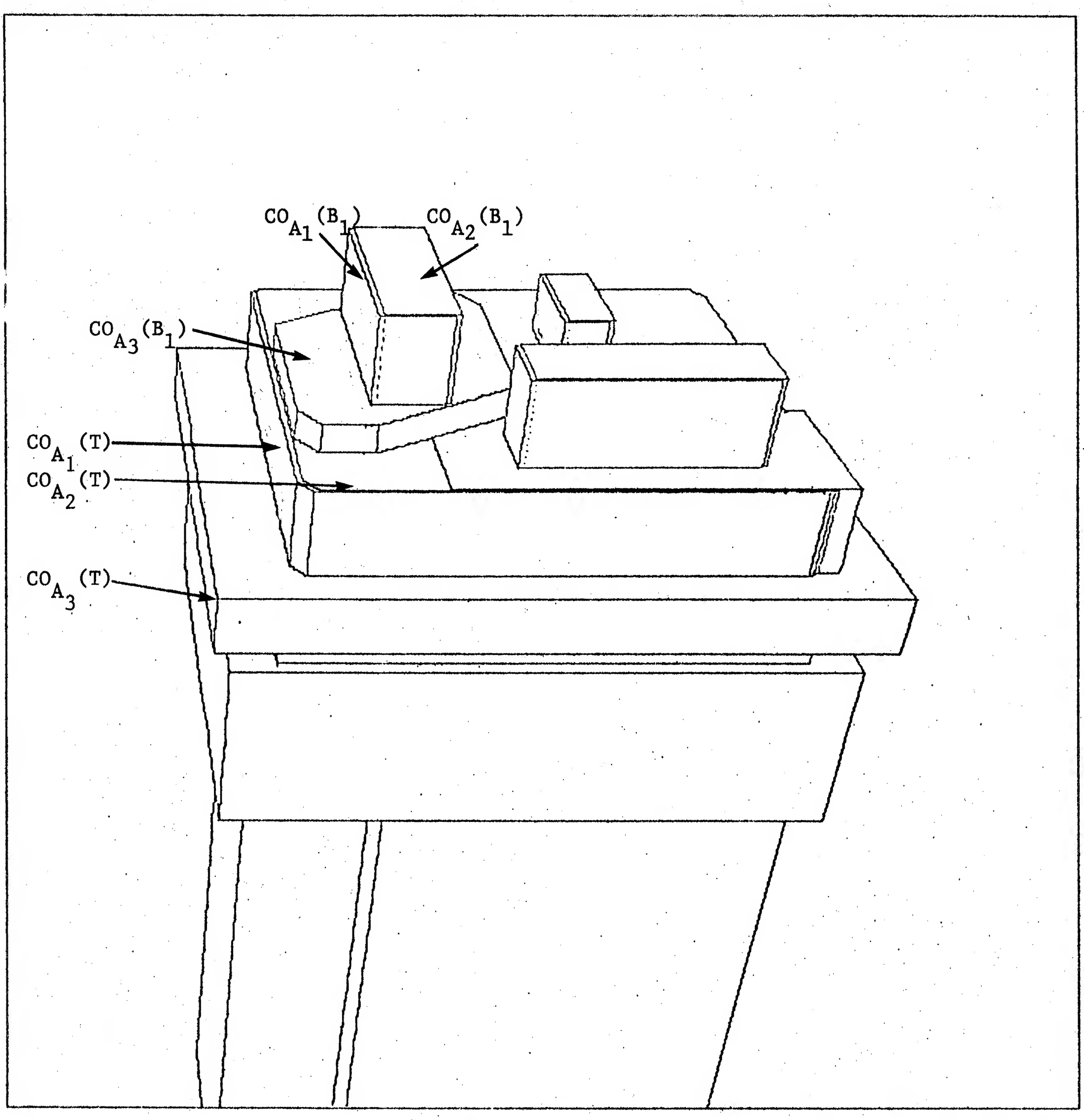


Figure 11b

START COS: The Cspace obstacles for the manipulator in the start configuration.

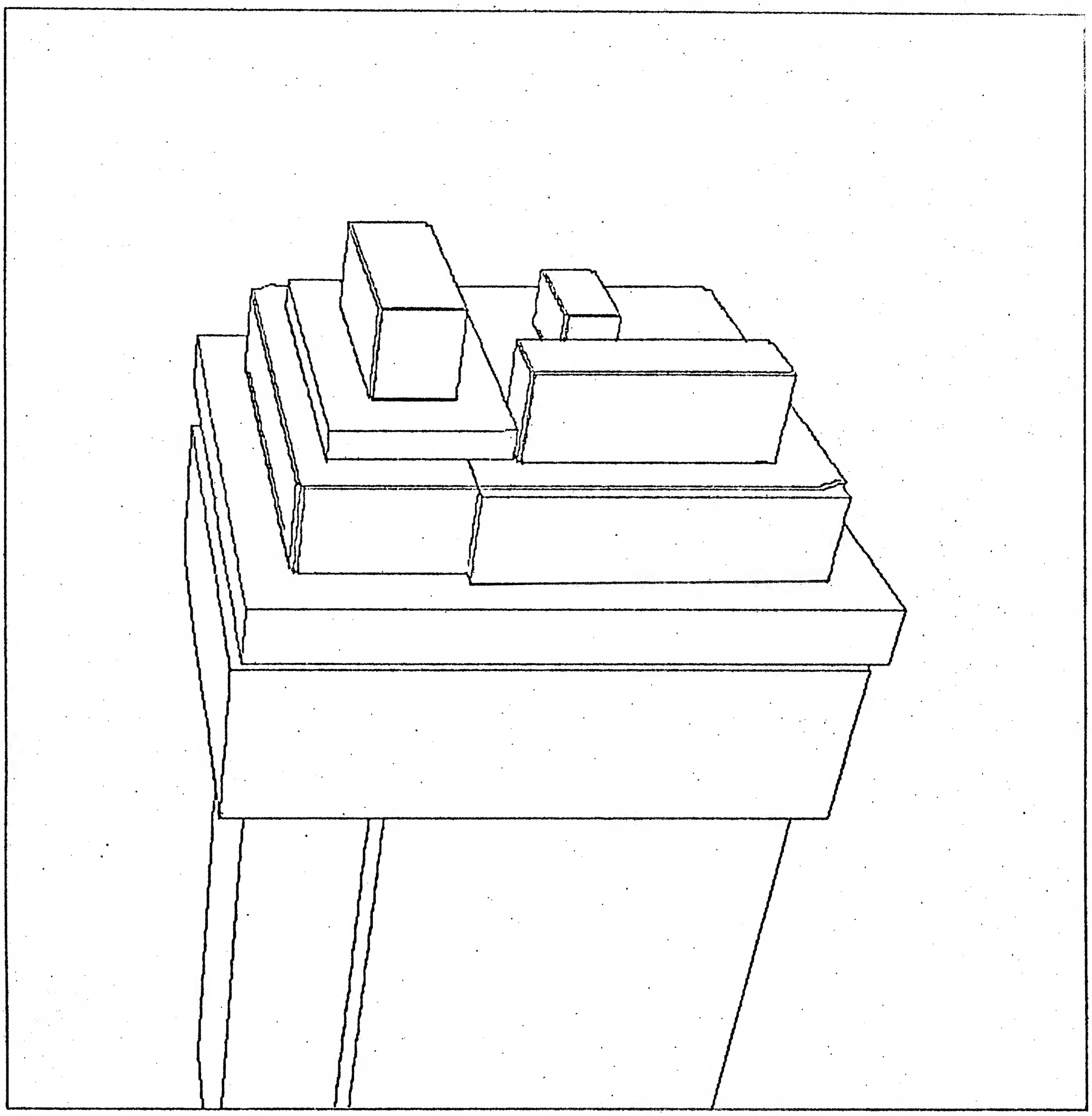


Figure 11c

GOAL COS: The Cspace obstacles for the manipulator in the goal configuration.

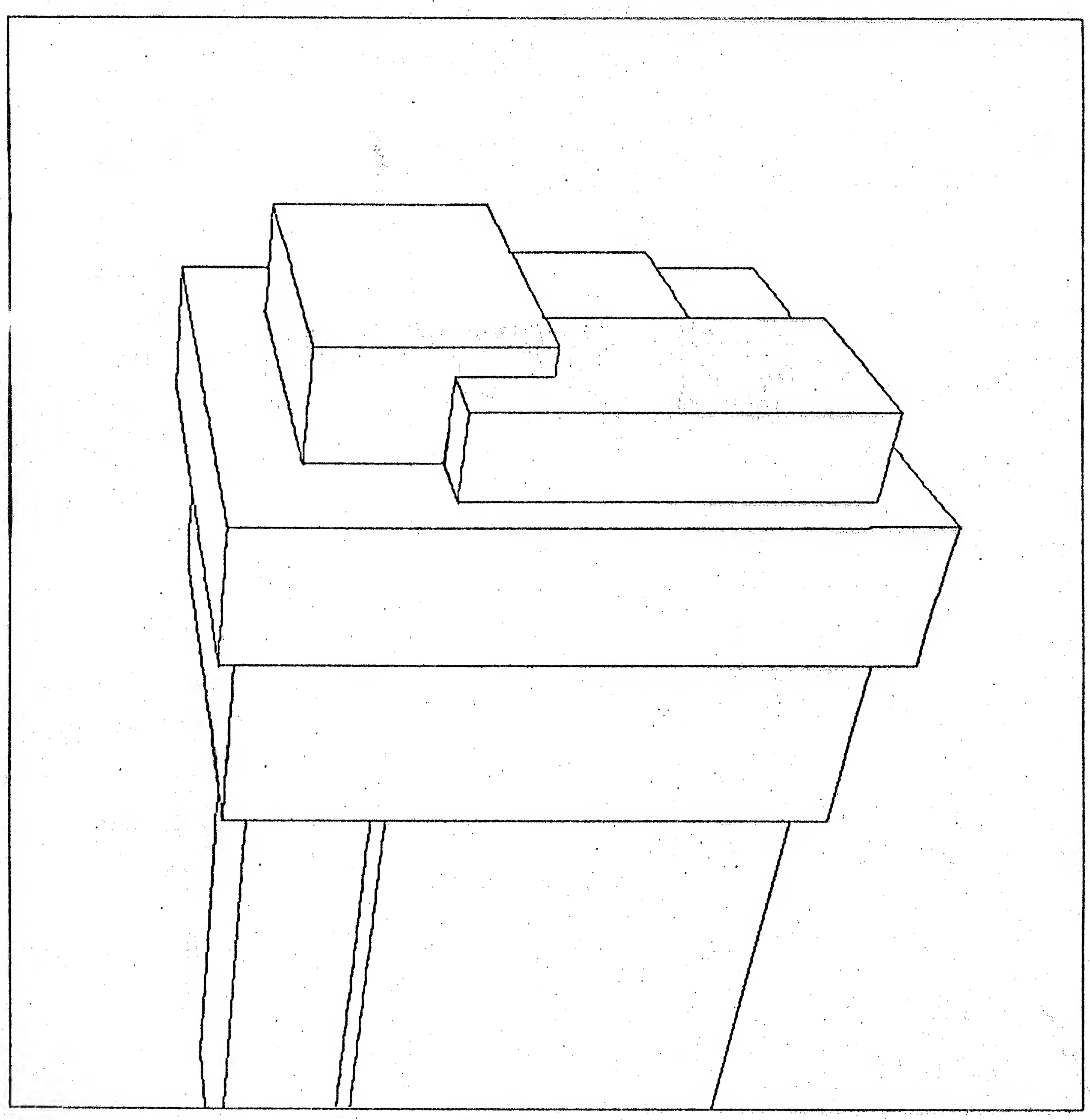


Figure 11d

The Cspace obstacles for the swept volume of the manipulator over a range of configurations of the wrist.

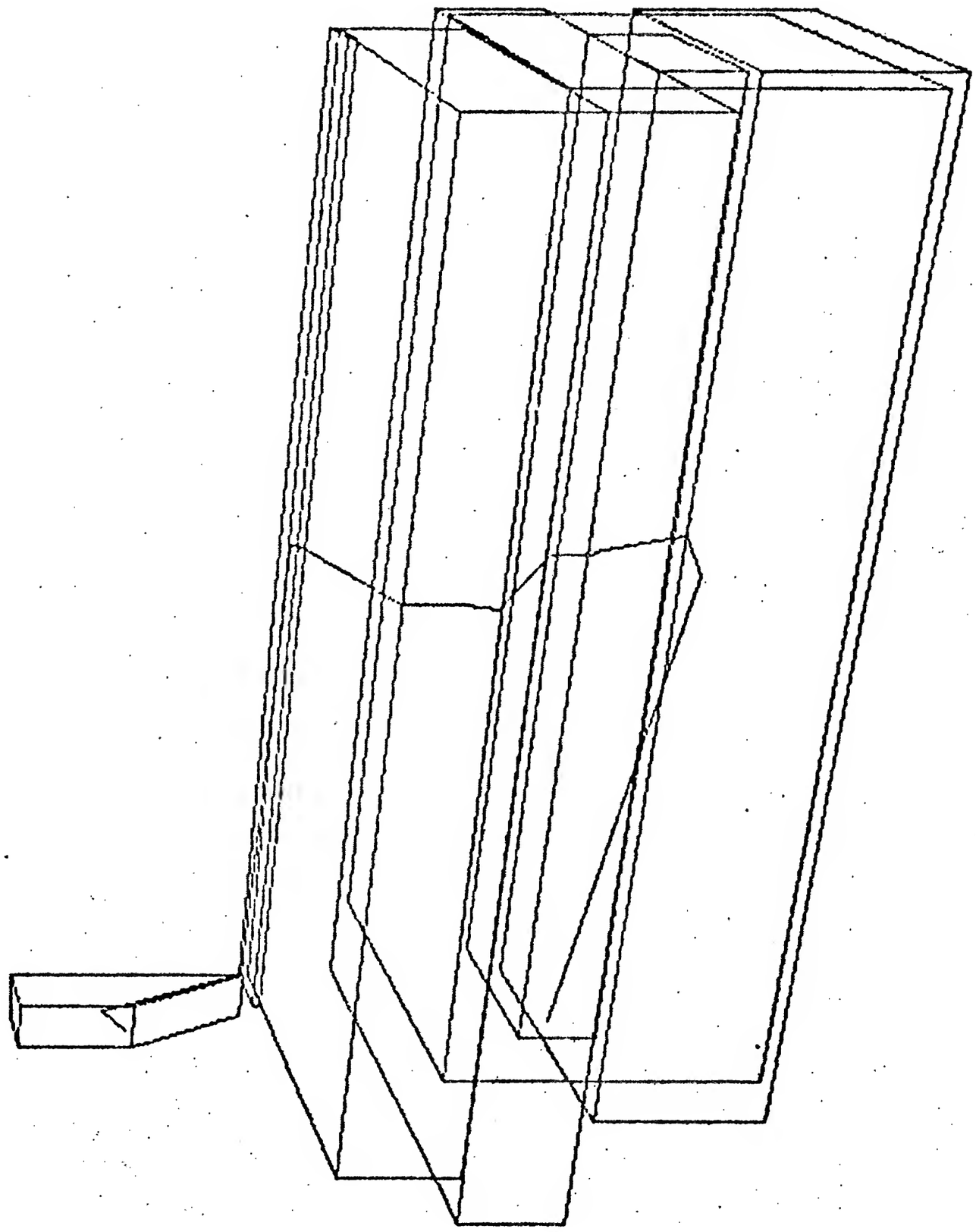


Figure 11e
The Cell Path with superimposed Line Path

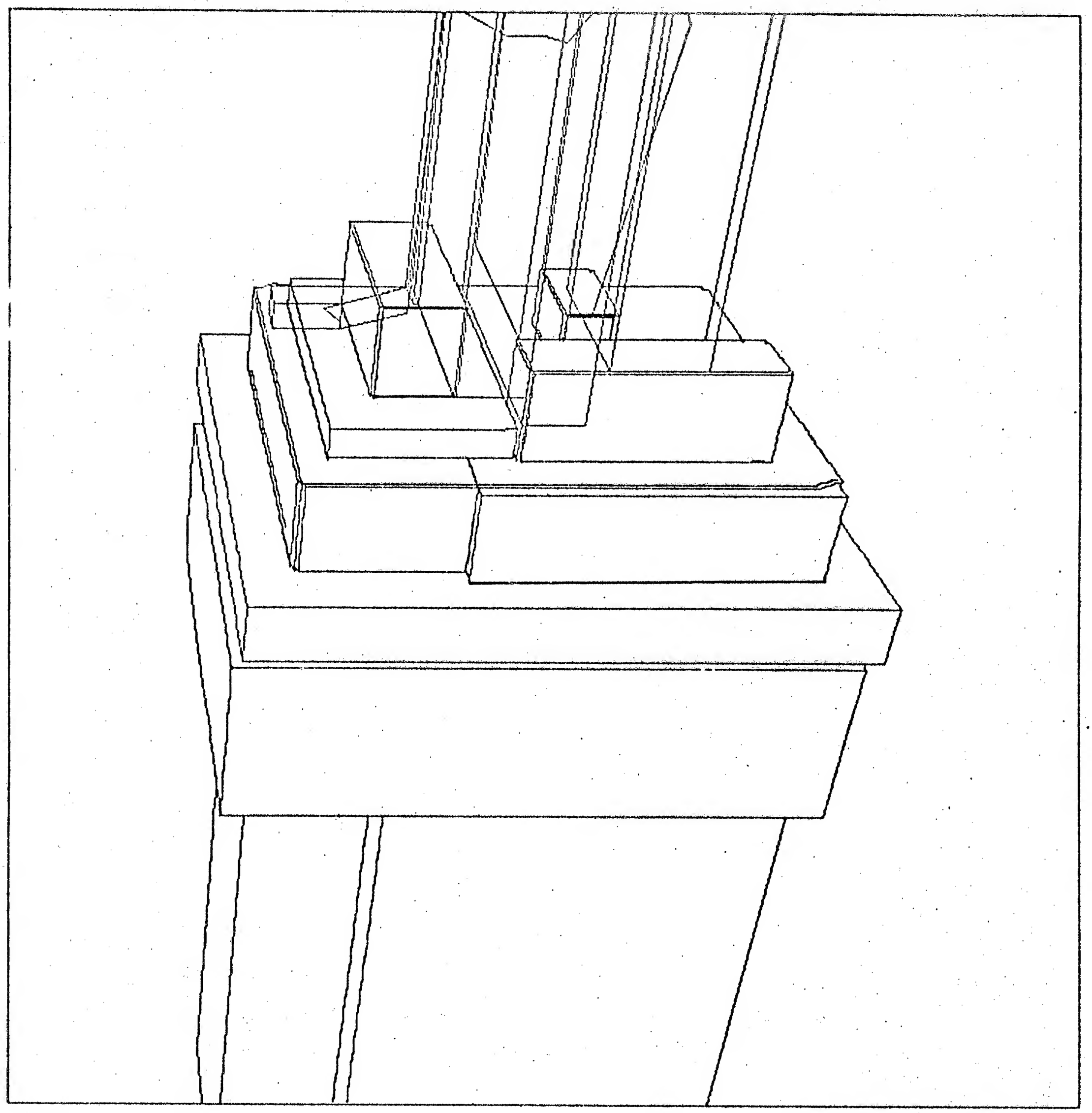


Figure 11f
The Cell Path and Line Path superimposed on GOAL COS

b. The *COS* for the start configuration. Each convex solid in the figure is a representation of $CO_{A_i}^{xyz}(B_j)$. Note that most of these $Cspace_A$ obstacles are rectangular solids, except for those arising from the interaction of the hand, A_3, with block B_1 and the fingers, A_1 and A_2, with the table. In these cases, the manipulator is so close to these obstacles that its configuration is *inside* the bounding rectangular solid for the configuration obstacles (In practice, the sides of the bounding rectangular solid are displaced outward by some small ϵ). This condition causes a detailed expansion to be carried out.

c. The *COS* for the goal configuration. In the goal configuration none of the obstacles needs to be expanded in detail.

d. The *COS* for one of the intermediate configuration ranges. This *COS* is defined for the manipulator's swept volume over a range of orientations of the wrist and hand. One bounding rectangular solid, A_1^*, approximates the swept volume of the hand and fingers, $A_1 \cup A_2 \cup A_3$. The solids A_4 and A_5 remain unchanged.

e. The cell path and the line path. This shows the cells from the various space representations that compose the cell path. One group of cells correspond to free space for the initial configuration, one large cell comes from the intermediate configuration (where the hand rotation takes place), and the last group of cells correspond to the final configuration. The line path shown goes through the centroid of each of the cells and also through the centroids of the intersection of adjacent cells on the path. Notice that because the cells are large, this path strategy produces paths that move too far from the obstacles. This could be overcome by subdividing the cells before finding the line path.

f. The cell path superimposed on the start *COS*. This shows the relative placing of the free cells relative to the obstacles.

9. Choosing Grasp Configurations

The preceding sections have discussed the problem of finding safe paths for the manipulator; this is only part of the "pick and place" synthesis problem. The major remaining problem is choosing

a grasp configuration on the part, P. For simple parts and non-cluttered environments, grasping is amenable to simple ad-hoc solutions. As a step in the solution of this problem, we deal here with choosing grasping configurations for relatively simple parts in cluttered environments. In this section, a *Cspace* approach to this problem is described, although no implementation of this approach to grasping currently exists.

The grasping problem is related to the Findspace problem introduced in Section 3, insofar as it involves choosing a safe configuration among a set of obstacles. But, there are additional constraints on the choice, for example:

1. the manipulator's fingers must be in contact with P,
2. the configuration must be reachable, and
3. P must be stable in the manipulator's hand, i.e. it will not slip in the hand during a motion.

The first two conditions, contact and reachability, reflect additional geometric constraints on the solution to the Findspace problem. The third condition, stability, reflects aspects of grasping beyond the purely geometric. Stability will be briefly discussed later in the section.

The approach to grasping described here is based on the one described in [25] and [26]. The basic idea is to build an explicit description of the set of configurations of the manipulator A for which the inside of the manipulator's fingers are in contact with specified surfaces of P. This set of configurations is some subset of $CO_A(P)$, call it G. Feasible grasp configurations are those in G, that do not cause any collisions with other objects in the workspace, i.e. that are outside all of the $CO_A(B_j)$. In this section, the details of this approach are discussed. We make the following simplifying assumptions:

1. The manipulator is cartesian and its hand is a parallel jaw, i.e. two parallel fingers that move along their common normal.
2. Only parallel planar surfaces, whose distance from each other is less than the maximum finger opening, are candidates for grasping. These are known as *grasp surfaces.*

These assumptions simplify the method for identifying feasible grasp configurations, while suggesting its usefulness and providing the foundation for a more general approach.

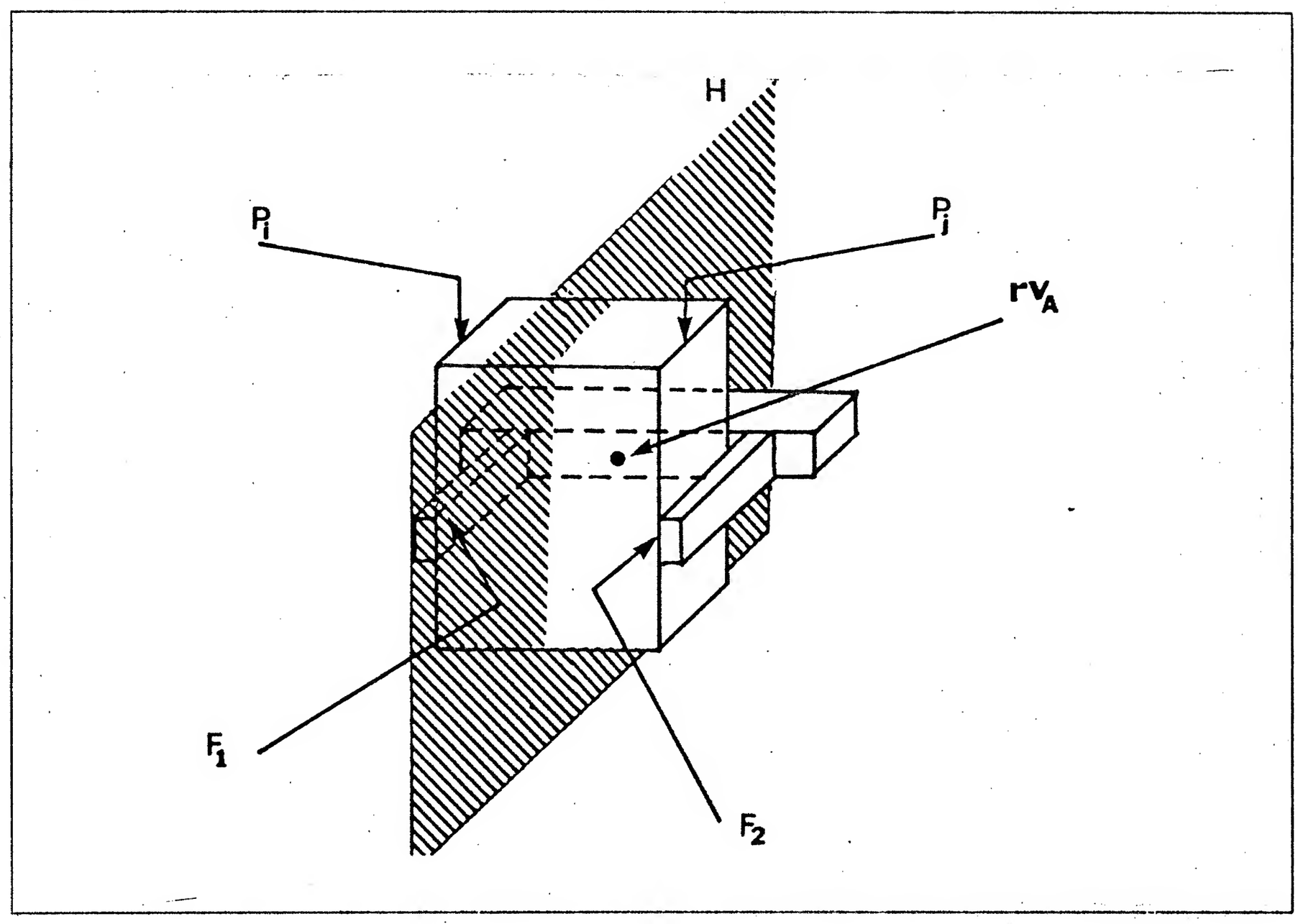


Figure 12. The definitions of P_i, P_j, F_1, F_2, and H used in choosing grasping configurations.

9.1. Feasible Grasp Configurations

Let P_i and P_j be the parallel faces[12] of P to be grasped, and F_1 and F_2 be the inside faces of the manipulator's fingers, Figure 12. Under the two assumptions stated above, when A grasps P, F_1 and F_2 are coplanar with P_i and P_j respectively. Under these conditions, the legal (x, y, z) positions of rv_A are restricted to some plane H that is parallel to P_i and P_j. Let $G_A(P_i, P_j)$ be the set of configurations of A for which rv_A is in H and for which P_i, P_j, F_1, and F_2 are mutually parallel. Note that $G_A(P_i, P_j)$ represents those positions and orientations where A could be when grasping P_i and P_j, without specifying the distance between the fingers. $G_A(P_i, P_j)$ is called the *grasp set* for P_i and P_j.

Note that not all the configurations in $G_A(P_i, P_j)$ are feasible grasp configurations, either because

[12]Note that objects in the current implementation are modelled as unions of convex polyhedra. Convex polyhedra are defined as the intersection of a finite number of half-spaces, where each half-space is bounded by a plane. The portion of each bounding plane on the boundary of the polyhedron is a convex polygon, known as a *face* of the object.

the fingers are not in contact with the grasp surfaces or because the manipulator configuration causes a collision with some other object. Therefore, we must impose two additional restrictions:

1. The internal faces of the fingers must overlap the grasp surfaces.
2. The manipulator must not collide with any other object in the workspace, i.e. the B_j.

With these restrictions on the configurations in the grasp set, we obtain the set of feasible grasp configurations, called a *feasible grasp set* and denoted $FG_A(P_i, P_j)$.

Define the configurations of F_1 and F_2 to correspond to those of the manipulator, i.e. each position and orientation of these faces is characterized by the manipulator configuration which would place them there. From these definitions it follows that $CO_{F_1}(P_i)$ is the set of those configurations of A for which the F_1 is in contact with P_i. Furthermore, $CO_{F_1}(P_i) \cap G_A(P_i, P_j)$ are those configurations for which the finger is in surface-surface contact with P_i. Therefore, it follows that

$$FG_A(P_i, P_j) = (CO_{F_1}(P_i) \cap CO_{F_2}(P_j) \cap G_A(P_i, P_j)) - \bigcup_j CO_A(B_j)$$

In this definition, we must let P be one of the B_j, say B_p, so as to avoid collisions with P while approaching a grasp configuration, but we must also allow A to contact P on the grasp surfaces. The answer is to add a slight displacement inward to P_i and P_j, when computing $CO_A(B_p)$, while using the original definition in the computation of $CO_{F_1}(P_i)$ and $CO_{F_2}(P_j)$.

The feasible grasp set, as defined above, is a volume in a six-dimensional $Cspace_A$. We do not have algorithms for computing this volume exactly. The algorithms of Section 3 serve only to compute slice projections of the $Cspace_A$ obstacles. It is clear that the same must be done for the feasible grasp set, namely computing its slice projection for some range of orientations. Such a slice would be the set of (x, y, z) positions of A that, for some range of orientations of A, are in contact with P, but outside all of the B_j. Presumably, this requires using the slice projections of $CO_{F_1}(P_i)$, $CO_{F_2}(P_j)$, and the $CO_A(B_j)$. A problem arises when trying to do this, because slice projections were defined over simple orientation ranges of the cartesian manipulator's wrist defined in Section 5. These ranges are not, in general compatible with the ranges of orientations that define $G_A(P_i, P_j)$. For a position of rv_A on H, only a small range of orientations will result in configurations that are in $G_A(P_i, P_j)$, yet for that position to be in a slice of $FG_A(P_i, P_j)$ it must be the case that no orientation within the slice's

defining range causes a collision. Therefore, few, if any, configurations in the grasp set will be feasible grasp configurations.

The solution to this problem is simply to define a new set of slices whose orientation ranges are subsets of the orientation ranges in $G_A(P_i, P_j)$. Note that a configuration in such a slice already satisfies the orientation constraints of the grasp set. Therefore, only the position constraints, i.e. that the (x, y, z) position be in H, need to be enforced to obtain the intersection of a *Cspace* obstacle in that slice with the grasp set. This removes the need of computing the complete representation of the obstacles, while simultaneously avoiding the problems introduced by irrelevant orientations.

Computing the obstacle slices for orientations in the grasp set requires being able to compute the swept volume of the manipulator over orientation ranges that are not the simple ranges of joint angles defined in Section 5. Let R be the set of orientations in the grasp set that define a slice and denote the swept volume of A over R as $A[R]$. Algorithms for approximating the swept volume over these ranges can be based on the simple approach described in the Appendix. The important constraint on the approximation to $A[R]$ is that it does not intersect the grasp surfaces for positions of rv_A on H.

In addition to the manipulator displacing and rotating, the manipulator's fingers may move perpendicular to the grasp surfaces. This additional degree of freedom has not been discussed above. In fact, it poses no additional problems; the motion of the fingers can be treated, via slice projection, uniformly with rotation. This simply requires including the space swept out by the fingers during closing, in the swept volume used to define slices of the $CO_A(B_j)$.

9.2. Overlap of Finger and Surface

The approach described above deals adequately with the $CO_A(B_j)$ in the definition of feasible grasp set, but is less succesful in dealing with $CO_{F_1}(P_i)$ and $CO_{F_2}(P_j)$. The reason for this is that a position in the slice projection of $CO_{F_1}(P_i)$ simply indicates that for <u>some</u> orientation of A in the slice, the finger is in contact with P_i. What is required instead is the set of positions which for <u>all</u> orientations of A in the slice, there is contact. In fact, we would like to guarantee that the area of contact between the fingers and the grasp faces always exceeds some fixed area. How this may be accomplished is discussed below.

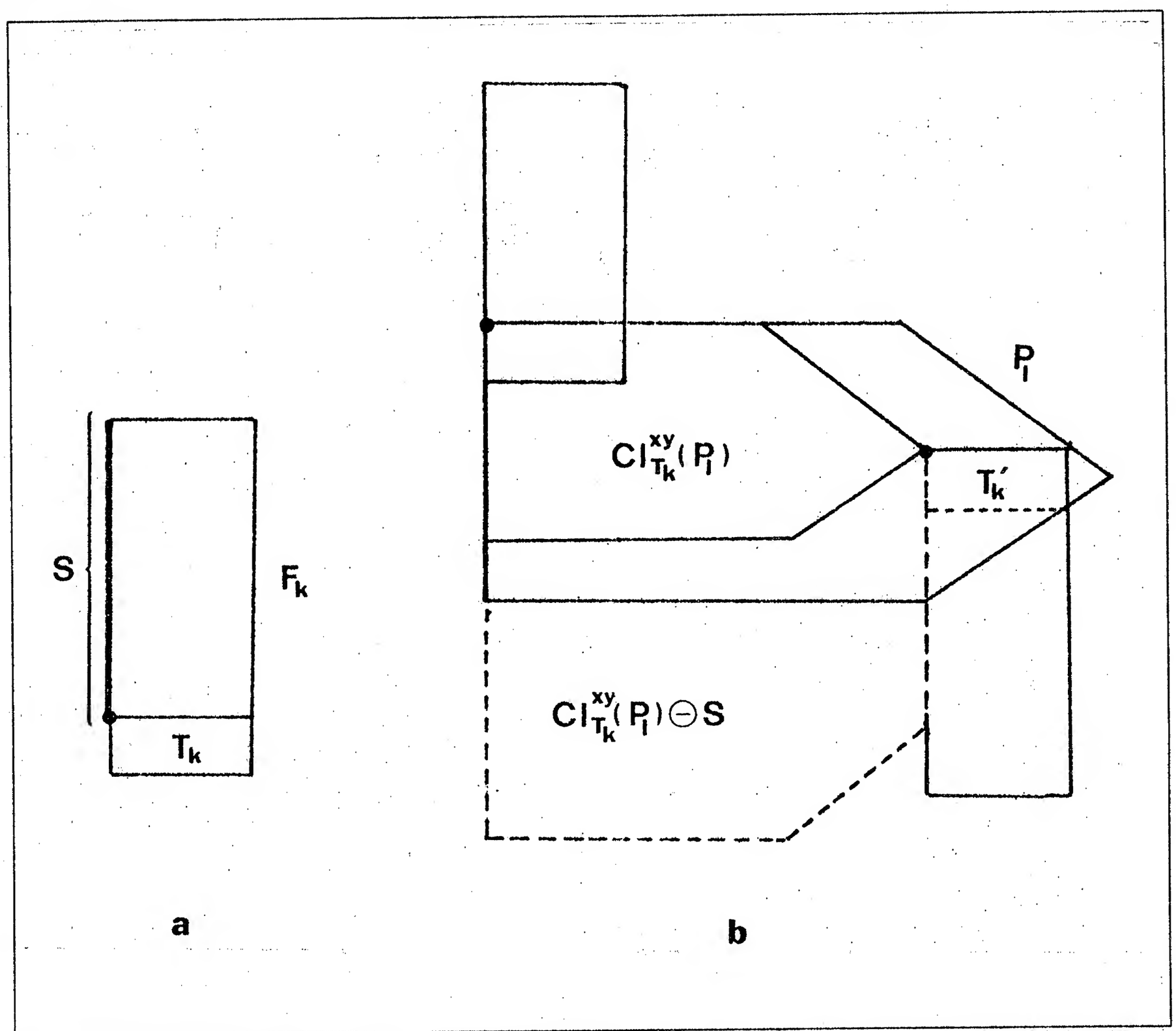


Figure 13. Defining the configurations of A for which F overlaps P. (a) Illustration of the definition of T_k and S. (b) Illustration of $CI^{xyz}_{T_k}(P_l) \ominus S$, with two positions of F_k's reference vertex (indicated by the small circles) showing the area of overlap includes an area of the form $T_k \oplus s$, for some $s \in S$.

Let F_k and P_l be, respectively, a finger surface and the corresponding grasp surface. We define T_k to be a small strip at the tip of F_k, such that $F_k = T_k \oplus S$, where S is the set of points along a line segment, as shown in Figure 13. Again, we assume that the configurations of T_k correspond to those of F_k (and therefore A). Assume A is in some configuration $c \in G_A(P_i, P_j)$, so that F_k and P_l are coplanar, then $CI^{xyz}_{T_k}(P_l)$ is the set of (x, y, z) configurations of T_k, and therefore of F_k and A, for which $F_k \cap P_l \supseteq T_k$. But, we do not want to restrict the overlap between F_k and P_l to be at the fingertip; instead, we want the area of overlap to include some area T'_k, obtainable by translating T_k along S, i.e. $T'_k = T_k \oplus \{s\}$, with $s \in S$. It is easy to show that

$$CI^{xyz}_{T_k}(P_l) \ominus S = \{\, c \mid \exists s \in S : P_l \cap (F_k)_c \supseteq (T_k)_c \oplus \{s\} \,\}$$

Therefore, this is the desired set of configurations, see Figure 13. This result can be applied to compute the slices needed for the feasible grasp set. If R is the orientation range defining the slice, then $CI^{xyz}_{T_k[R]}(P_l) \ominus S[R]$ represents the set of (x, y, z) configurations that, for orientations in R, guarantee that the contact between F_k and P_l includes T_k. Note that this approach can be generalized to any S and T_k such that $F_k = T_k \oplus S$; as T_k becomes smaller and approaches a point, then S approaches F_k.

9.3. Safety at the Destination

So far, the definition of $FG_A(P_i, P_j)$ only embodies constraints relating to safety at P's initial configuration, however a grasp configuration must also be safe at P's final configuration. Clearly, another feasible grasp set can be computed at P's final configuration, say $FG_A(P'_i, P'_j)$ where the primed faces indicate the faces at their final configuration. But, these two feasible grasp sets cannot be intersected to obtain those grasp configurations that are safe for both configurations of P, because a grasp configuration corresponds to different manipulator configurations at each different configuration of P. What is needed is a way of defining those grasp configurations in P's initial configuration that would lead to a collision when P is in its final configuration, Figure 14.

A grasp configuration establishes a fixed relationship between the fingers and the grasped part, P. Let the final configuration of P be obtained by a displacemnet consisting of a translation t and a rotation r, indicated by $D_{t,r}(P)$. Clearly, any set of positions X bears the same relationship to $D_{t,r}(P)$ as $D^{-1}_{t,r}(X)$ bears to P. Therefore, if $CO^{xyz}_A(B_j)$ is a set of positions of A which cause collisions at P's final configuration, then $D^{-1}_{t,r}(CO^{xyz}_A(B_j))$ represent infeasible grasp configurations, Figure 14. This result also holds for swept volumes of A, therefore it may be used to ensure safety at the destination in the definition of feasible grasp sets.

9.4. Computing the Feasible Grasp Set

The discussion in the preceding subsections is summarized in the following definition of feasible grasp set, for some range of orientation in the grasp set. We denote this orientation range as R, and let R' denote the same orientation range as R relative to P, but at P's destination. We also let (t, r) be

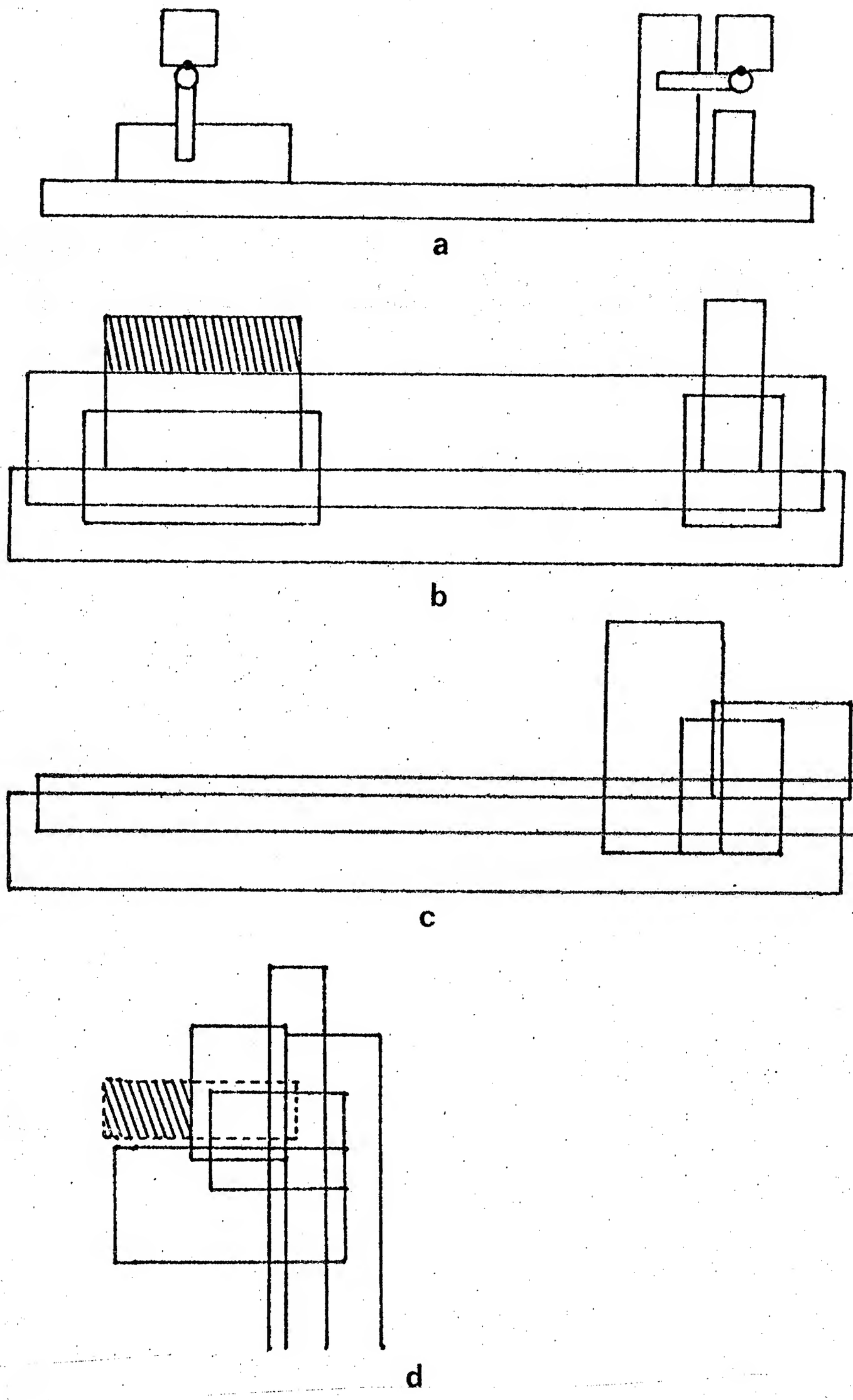


Figure 14. (a) A side view of a manipulator hand, composed of a finger and a "palm", holding P at the initial and final configuration. (b) In the initial configuration, the shaded area represents $CO_{F_1}^{xyz}(P_i) - \bigcup_j CO_A^{xyz}(B_j)$, i.e. the feasible grasp configurations for A, considering only safety at the origin and letting T be a point. (c) The $CO_A^{xyz}(B_j)$ for the final configuration of A and P. (d) The shaded area represents $CO_{F_1}^{xyz}(P_i) - \bigcup_j CO_A^{xyz}(B_j) \cup D_{t,r}^{-1}(CO_A^{xyz}(B_j))$, which is the feasible grasp set that takes into account safety at the destination.

the displacement between the initial and final configurations of P. Then, the feasible grasp set, for the orientation range R and displacement (t, r), is:

$$FG^{xyz}_{A[R]}(P_i, P_j) = ((CI^{xyz}_{T_1[R]}(P_i) \cap CI^{xyz}_{T_2[R]}(P_j)) \ominus S[R]) - \bigcup_j CO^{xyz}_{A[R]}(B_j) \cup D^{-1}_{t,r}(CO^{xyz}_{A[R']}(B_j))$$

All of the elements in this definition can be computed using the CO^{xyz} algorithm of Section 3.2 and a swept volume algorithm.

9.5. Approach and Departure

Configurations in the feasible grasp set, as defined above, are guaranteed to be safe both at P's initial and final configuration. While these conditions are sufficient in most situations, they do not guarantee that the feasible grasp configurations can be used during a "pick and place" operation. For a feasible grasp configuration to be a *legal* grasp configuration, it must allow the manipulator to reach and depart P's initial and final configurations. Summarizing, the following conditions must hold for a legal grasp configuration:

1. It must be possible to reach it from the initial configuration of the manipulator.
2. It must be possible to remove P from its initial configuration safely.
3. It must be possible to reach the P's final configuration with P held in the hand.
4. It must be possible to withdraw the manipulator from P's final configuration.

The Findpath algorithm described in the preceeding sections can be extended to deal with the problem of choosing a grasping configuration that is reachable from the manipulator's initial configuration. As we saw above, the feasible grasp configurations, over some range of orientations, are those within some specified volume of $Cspace_A$, but outside the slice projections of suitably defined $Cspace_A$ obstacles. Hence, they are equivalent to the slices, $COS[c, c']$ of Section 6.2. Therefore, a free space representation for the feasible grasp configurations can be constructed and the resulting free cells linked in the Free Space Graph. The feasible grasp configurations for alternative grasp surfaces can also be linked into the graph. In the resulting FSG, any path from the cell containing the origin to a cell containing a feasible grasp configuration shows that this grasp configuration may

be reached from the origin. The path searching process must be modified to search for any cell which contains a suitable grasp configuration, rather than searching for a particular cell containing the destination.

Similarly, departure from the origin and approach to the destination could be handled by testing whether the destination is reachable, using the FSG constructed as above. The difference is that now the hand is holding P, therefore the polyhedral description of P must be treated as if it were part of the manipulator. This requires adding a new set of $Cspace_A$ obstacles, arising from the interaction of P and the objects in the workspace, to the ones already computed for the manipulator. This is entirely analogous to modifying the description of the manipulator, which is already modelled as a union of convex solids. But, the geometric relationships between P and the A_i are determined by the grasp configuration, which has several degrees of freedom. The problem can be approached by treating these additional degrees of freedom, via slice projection, just as the wrist rotations were treated. This approach imposes a great cost in additional computation. A simpler, though less general, technique is to use heuristics in choosing a feasible grasp configuration and then test, via the path search process, whether that grasp configuration permits departure. If it does not, a new configuration might be chosen and the process repeated. This approach would be not be adequate for very cluttered environments or situations involving parts mating at the destination. In such environments an approach based on slice projection would also be susceptible to failure. Further research is needed in this area.

9.6. Stability in Grasping

We have thus far not considered the issue of stability of the feasible grasp point. An adequate treatment of stability in grasping is not yet available, although some promising approaches exist [5]. The techniques described in this section can be used to implement two simple grasping heuristics, which work adequately when (1) the manipulator hand is made up of rigid fingers, (2) the object to be grasped, P, is small relative to the manipulator hand and (3) parts mating effects are ignored. The two heuristics are:

1. Ensure at least a minimum contact area of the fingers with the grasp surfaces. The amount of overlap should depend on object properties such as weight and surface smoothness.

2. The perpendicular projection of P's center of mass should be near to $F_1 \cap P_i$ and $F_2 \cap P_j$.

The implementation of the contact area heuristic was discussed above, Section 9.2. The center of mass heuristic can be implemented by giving preference to grasp surfaces for which the center of mass, projected onto the plane containing P_i, falls within P_i and similarly for P_j. Furthermore, for specified grasp surfaces, the choice among legal grasp configurations should minimize the distance of the prjection of the center of mass to the area of overlap between finger and grasp surface.

These heuristics, though adequate for many tasks, are not a substitute for a general theory of stability in grasping. This remains one of the most interesting open problems in robotics.

10. The Effect of Uncertainty

In the preceeding sections we have assumed that:

1. the configuration of all the objects is known exactly, and
2. the configuration of the manipulator can be controlled exactly.

Both of these assumptions are only approximations to reality. In practice, configurations can only be known to within some uncertainty. Both of these sources of uncertainty affect what manipulator motions are safe.

10.1. Modelling Worst-Case Uncertainty in $Cspace_A$

In $Cspace_A$, the two sources of uncertainty have similar effects, i.e. modifying the shape of the $Cspace_A$ obstacles. This section deals with techniques for taking into account these effects. The following notation is useful in the discussion. Let $e = (\beta_i) = (\beta_1, \ldots, \beta_n) \in \Re^n$ and similarly, let configurations be $(\gamma_i) = (\gamma_1, \ldots, \gamma_n) \in \Re^n$. The index set $\{1, \ldots, n\}$ will be referred to as I; let $K \subseteq I$. The set $U_K(e)$ denotes the set of configurations in $Cspace_A$ whose K-parameters are less than the absolute value of the corresponding parameter of e.

$$(\gamma_i) \in U_K(e) \Leftrightarrow \begin{cases} -\beta_i \leq \gamma_i \leq \beta_i, & \text{if } i \in K \\ \gamma_i = 0, & \text{otherwise} \end{cases}$$

Uncertainty in the configuration of A in $Cspace_A$ can be represented as a region around its *nominal configuration*, c; within this region are all the configurations that A may be in. Simple regions can be characterized by $\{c\} \oplus U_K(e_A)$. Assume that $(A)_a \cap B \neq \emptyset$, i.e that $a \in CO_A(B)$. Any nominal configuration a' such that $a' + x = a$, for $x \in U_K(e_A)$, should also belong to $CO_A(B)$. This means that under uncertainty of A, $CO_A(B)$ should be replaced with $CO_A(B) \ominus U_K(e_A)$. In practice, we do not ever compute $CO_A(B)$; rather, we compute slice projections of it using the swept volume of A over ranges of orientation parameters, R. Therefore, the orientation and translation uncertainty must be treated separately. Orientation uncertainty affects the definition of the manipulator's swept volume. For example, to compute a slice with parameters $[c, c']_R$, the swept volume $A[c-e_A, c'+e_A]_R$ is used in place of $A[c, c']_R$. The effect of the uncertainty in the translation parameters, T, can be computed as indicated in Section 5.1, using the $CO_A^{xyz}(B)$ algorithm.

The worst-case effect on $CO_A(B_j)$ of uncertainty in the configuration of the B_j, can be modelled by replacing B with the swept volume of B over the uncertainty range. Alternatively, if the uncertainty in the configuration of B_j can be approximated by an uncertainty in translation[13] , $U_T(e_{B_j})$ then the uncertainty of A and B can be combined into a single uncertainty[14] and treated as the uncertainty of A. If T is the set of indices for translation parameters, then the combined uncertainty is:

$$U_T(e'_A) = U_T(e_{B_j}) \ominus U_T(e_A)$$

10.2. The Effect of Uncertainty on "Pick and Place" Synthesis

The presence of uncertainty significantly affects manipulator programming in general and the synthesis of "pick and place" motions in particular. One approach to planning motions in the presence of uncertainty is to plan paths that are safe under the worst case uncertainty, i.e. paths outside the expanded $Cspace_A$ obstacles defined above. This approach rules out most operations that involve moving near objects, e.g. grasping. Another approach is to assume that uncertainty does not significantly affect the outcome of most operations and to plan motions assuming nominal

[13]This can be done by defining a new translation uncertainty such that the swept volume over this range of positions will contain the swept volume over the original uncertainty range.

[14]This assumes that the translation space of the manipulator is the same as that of the objects in the workspace, which is true for cartesian manipulators.

configurations. A compromise position is to redefine the "pick and place" synthesis problem so as to isolate those operations that are most susceptible to uncertainty from those others where uncertainty plays a relatively minor role. The latter can be addressed by the techniques outlined in this paper, the former require a different approach. One possible re-definition of the "pick and place" problem is the following:

1. Find a nominal grasp configuration assuming that there is no uncertainty.

2. Identify a *grasp approach configuration*, a configuration that can be shown to be safe under worst-case uncertainty estimates for object and manipulator configuration.

3. Identify a *grasp deproach configuration*, a configuration which is safe for the manipulator grasping the part, given the uncertainty in the part's configuration after grasping and the uncertainty in configurations of nearby objects.

4. Compute a path, from the manipulator's initial configuration to the grasp approach configuration, assuming worst-case uncertainty.

5. Identify a *destination approach configuration*, a configuration which is safe for the manipulator holding the object, given the uncertainty in the grasp configuration and the uncertainty of nearby objects.

6. Compute a safe path from the grasp deproach configuration to the destination approach configuration for the manipulator and the grasped part, also assuming wort-case uncertainty.

7. Identify a *destination depproach configuration*, a configuration which is safe for the manipulator, given the uncertainty of nearby objects.

This formulation of the synthesis problem factors out the problems of approaching and deproaching both the nominal grasp configuration and the destination. For both of these problems, the use of sensory information to identify the actual state of the task and to accomodate to it is important [25] [30] [44]. When the uncertainty is small, the problem can be dealt with by *ad hoc* methods, e.g. opening the fingers very wide and relying on the grasping action to place the object and/or the manipulator in approximately the correct orientation [19]. The general problem of planning manipulator operations that are robust in the face of uncertainty is an important problem [30], but

beyond the scope of this paper.

11. Summary

This paper has presented an approach to the central geometric problems underlying the synthesis of "pick and place" motions for cartesian manipulators. The key technique in the approach is the use of explicit polyhedral representations of the configuration constraints on the manipulator. This representation permits the use of simple and powerful geometric operations to solve problems involving safe motions of the manipulator. In particular, the problems of finding grasp configurations and safe paths in the absence of uncertainty.

The concepts of Configuration Space and Configuration Space Obstacle have played a central role in the approach to gross motion synthesis developed here. Similar concepts play an important role in the approach to compliant motion synthesis described in [30]. These concepts have also proven useful in other geometric applications [1] [2] [4] [45].

Acknowledgements

All of this paper, in particular the section on choosing grasp configurations, has benefited greatly from the criticism, insights, and suggestions of my colleague, Matt Mason. I would also like to thank Mike Brady, John Hollerbach, Berthold Horn, and Patrick Winston for reading drafts of this paper and, in general, for their help and encouragement.

References

[1] **Adamowicz, M.** The optimum two-dimensional allocation of irregular, multiple-connected shapes with linear, logical and geometric constraints, PhD Thesis, Department of Electrical Engineering, New York University, 1970.

[2] **Adamowicz, M and Albano, A.** "Nesting two-dimensional shapes in rectangular modules," *Computer Aided Design* **8**, 1 (Jan 1976), 27–32.

[3] **Ahuja, N; Chien, R T; Yen, R and Bridwell, N.** "Interference Detection and Collision Avoidance Among Three Dimensional Objects," *First Annual National Conference on Artificial Intelligence* , Stanford University, August 1980.

[4] **Albano, A and Sapuppo, G.** "Optimal Allocation of Two-Dimensional Irregular Shapes Using Heuristic Search Methods," *IEEE Transactions on Systems, Man, and Cybernetics* **SMC-10**, 5 (May 1980), 242–248.

[5] **Asada, H.** "Studies in Prehension and Handling by Robot Hands with Elastic Fingers," University of Kyoto, 1979.

[6] **Baumgart, B G.** "Geometric Modelling for Computer Vision," Stanford Artificial Intelligence Laboratory, Memo 249, October 1974.

[7] **Boyse, J W.** "Interference Detection Among Solids and Surfaces," *Communications of the ACM* **22**, 1 (January 1979), 3–9.

[8] **Dyer, C R; Rosenfeld, A and Samet, H.** "Region Representation: Boundary Codes from Quadtrees," *Communications of the ACM* **23**, 3 (March 1980), 171–179.

[9] **Eastman, C E.** "Representations for Space Planning," *Communications of the ACM* **13**, 4 (April 1970), 242–250.

[10] **Finkel, R; Taylor, R; Bolles, R; Paul, R; and Feldman, J.** "AL, A Programming System for Automation," Stanford Artificial Intelligence Laboratory, AIM-177, Nov 1974.

[11] **Freeman, H.** "On the Packing of Arbitrary-Shaped Templates," *Second USA-Japan Computer Conference* , 1975, 102–107.

[12] **Giralt, G; Sobek, R and Chatila, R.** "A Multilevel Planning and Navigation System for a Mobile Robot," *Sixth International Joint Conference on Artificial Intelligence* , Tokyo, Japan, August 1979, 335–338.

[13] **Golden, B.** "Shortest Path Algorithms: A Comparison," *Operations Research* **24**, 6 (November 1976), 1164–1168.

[14] **Grunbaum, B.** *Convex Polytopes* , Wiley Interscience, New York, 1967.

[15] **Hart, P; Nilsson, N and Raphael, B.** "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transactions on System Science and Cybernetics* **SSC-4**, 2 (July 1968), 100–107.

[16] **Howden, W E.** "The Sofa Problem," *Computer Journal* **11**, 3 (November, 1968), 299–301.

[17] **Hunter, G M and Steiglitz, K.** "Linear Transformation of Pictures Represented by Quad Trees," *Computer Graphics and Image Processing* **10**, (1979), 289–296.

[18] **Hunter, G M and Steiglitz, K.** "Operations on Images Using Quad Trees," *IEEE Transactions on Pattern Analysis and Machine Intelligence* **PAMI-1**, 2 (April 1979), 145–153.

[19] **Inoue, H.** "Force Feedback in Precise Assembly Tasks," MIT Artificial Intelligence Laboratory, AIM-308, August 1974.

[20] **Klinger, A and Dyer, C R.** "Experiments on Picture Representation Using Regular Decomposition," *Computer Graphics and Image Processing* **5**, 1 (1976), 68–105.

[21] **Korn, G A and Korn, T M.** *Mathematical Handbook for Scientists and Engineers* , McGraw Hill, New York, 1968.

[22] **Larson, R C and Li, V O K.** "Finding Minimum Rectilinear Distance Paths in the Presence of Obstacles," MIT Operations Research Center, OR 088-79, May 1979.

[23] **Lewis, R A.** "Autonomous Manipulation on a Robot: Summary of Manipulator Software Functions," Jet Propulsion Laboratory, California Institute of Technology, TM 33-679, March 1974.

[24] **Lieberman, L and Wesley, M A.** "AUTOPASS: An Automatic Programming System for Computer Controlled Assembly," *IBM Journal of Research and Development* **21**, 4 (July 1977).

[25] **Lozano-Perez, T.** "The Design of a Mechanical Assembly System," MIT Artificial Intelligence Laboratory, TR-397, Dec 1976.

[26] **Lozano-Perez, T and Winston, P H.** "LAMA: A Language for Automatic Mechanical Assembly," *Fifth International Joint Conference on Artificial Intelligence* , Massachusetts Institute of Technology, August 1977, 710–716.

[27] **Lozano-Perez, T.** "Spatial Planning: A Configuration Space Approach," *IEEE Transactions on Computers* (To appear).

[28] **Lozano-Perez, T and Wesley, M A.** "An Algorithm for Planning Collision-Free Paths among Polyhedral Obstacles," *Communications of the ACM* **22**, 10 (October 1979), 560–570.

[29] **Marr, D and Nishihara H K.** "Representation and Recognition of the Spatial Organization of Three Dimensional Shapes," MIT Artificial Intelligence Laboratory, AIM-416, May 1977.

[30] **Mason, M T.** "Compliance and Force Control for Computer Controlled Manipulators," MIT

Artificial Intelligence Laboratory, TR-515, April 1979.

[31] **Mathur, G.** "The Grasp Planner," Department of Artificial Intelligence, University of Edinburgh, DAI Working Paper 1, 1974.

[32] **Moravec, H P.** "Visual Mapping by a Robot Rover," *Proceedings Sixth International Joint Conference on Artificial Intelligence*, Tokyo, Japan, August 1979.

[33] **Nilsson, N.** "A Mobile Automaton: An Application of Artificial Intelligence Techniques," *Proceedings International Joint Conference on Artificial Intelligence*, 1969, 509–520.

[34] **Park, W T.** "Minicomputer Software Organization for Control of Industrial Robots," *Joint Automatic Control Conference*, San Francisco, 1977.

[35] **Paul, R P.** "Modelling, Trajectory Calculation and Servoing of a Computer Controlled Arm," Sranford Artificial Intelligence Laboratory, AIM-177, November 1972.

[36] **Paul, R P.** "Manipulator Cartesian Path Control," *IEEE Transactions on Systems, Man, and Cybernetics* **SMC-9**, 11 (November 1979), 702–711.

[37] **Pfister, G.** "On Solving the FINDSPACE Problem, or How to Find Where Things Aren't," MIT Artificial Intelligence Laboratory, Working Paper 113, March 1973.

[38] **Popplestone, R J.** "Specifying Manipulation in Terms of Spatial Relationships," Department of Artificial Intelligence, University of Edinburgh, No. 117, June 1979.

[39] **Popplestone, R J; Ambler, A P and Bellos, I M.** "An Interpreter for a Language for Describing Assemblies," *Artificial Intelligence* **14**, 1 (1980), 79–107.

[40] **Popplestone, R J; Ambler, A P and Bellos, I M.** "RAPT: A Language for Describing Assemblies," *Industrial Robot* **5**, 3 (1978), 131–137.

[41] **Preparata, F and Hong, S.** "Convex Hulls of Finite Sets of Point in Two and Three Dimensions," *Communications of the ACM* **20**, 2 (Feb 1977), 87–93.

[42] **Reddy, D R and Rubin, S.** "Representation of Three-Dimensional Objects," Department of Computer Science, Carnegie-Mellon University, CMU-CS-78-113, April 1978.

[43] **Samet, H.** "Region Representation: Quadtrees from Boundary Codes," *Communications of the ACM* **23**, 3 (March 1980), 163–170.

[44] **Simunovic, S N.** "Force Information in Assembly Processes," *Fifth International Symposium on Industrial Robots*, September 1975.

[45] **Stoyan, Y G and Ponomarenko L D.** "A Rational Arrangement of Geometric Bodies in Automated Design Problems," *Engineering Cybernetics* **16**, 1 (January 1978).

[46] **Taylor, R.** "A Synthesis of Manipulator Control Programs from Task-Level Specifications," Stanford Artificial Intelligence Laboratory, AIM-282, July 1976.

[47] **Thompson, A M.** "The Navigation System of the JPL Robot," *Fifth International Joint Conference on Artificial Intelligence* , Massachusetts Institute of Technology, 1977.

[48] **Udupa, S.** "Collision Detection and Avoidance in Computer Controlled Manipulators," *Fifth International Joint Conference on Artificial Intelligence* , Massachusetts Institute of Technology, 1977.

[49] **Udupa, S.** Collision Detection and Avoidance in Computer Controlled Manipulators, PhD Thesis, Department of Electrical Engineering, California Institute of Technology, 1977.

[50] **Widdoes, C.** "A Heuristic Collision Avoider for the Stanford Robot Arm," Stanford Artificial Intelligence Laboratory, 1974.

[51] **Wingham, M.** Planning How to Grasp Objects in a Cluttered Environment, M. Phil Thesis, Department of Artificial Intelligence, University of Edinburgh, 1977.

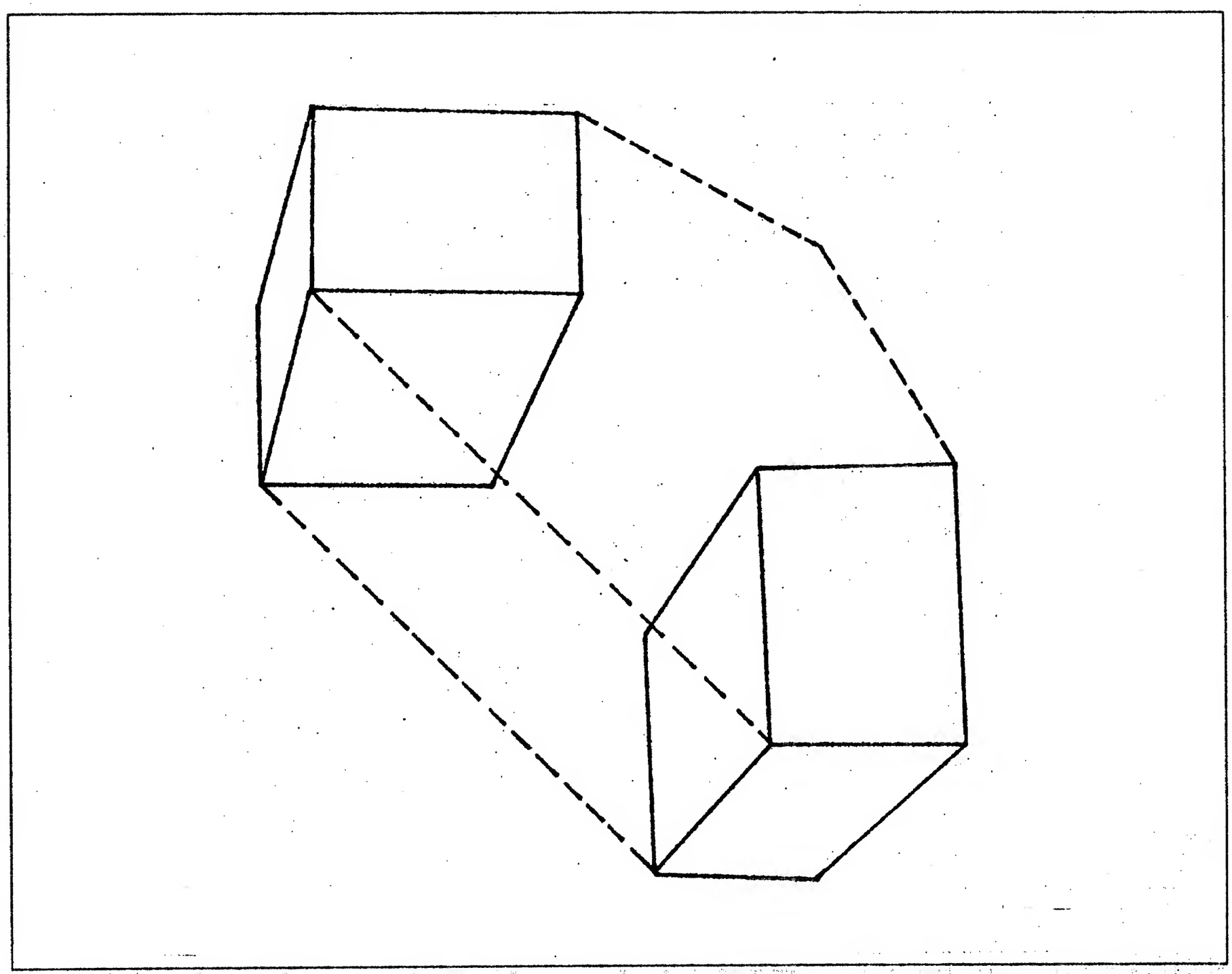


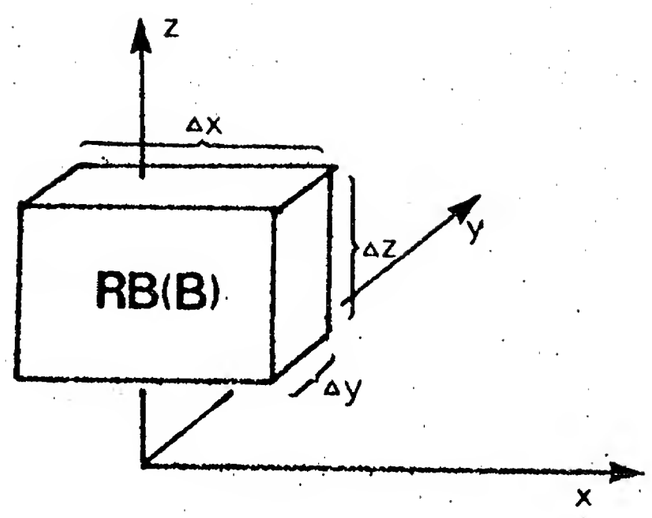
Figure 15. The WEDGE is a convex polyhedron used to approximate the volume swept out by a cuboid aligned with the coordinate axes, as it rotates around the z axis, assuming the z axis does not penetrate the cuboid.

Appendix 1. A Polyhedral Approximation for Swept Volume

The *swept volume* is the volume occupied by a polyhedron over a set of configurations, e.g. along some path. The swept volume over a range of translations can be computed using the CO^{xyz} algorithm. In this appendix, we will limit our attention to computing a simple polyhedral approximation to the swept volume for rotations of a polyhedron around an arbitrary axis. This method is included here for completeness, it is not the best polyhedral approximation to the swept volume.

The swept volume approximation described here returns a list of convex polyhedra of two types:

1. CYLINDER — a polyhedral approximation to a right circular cylinder.



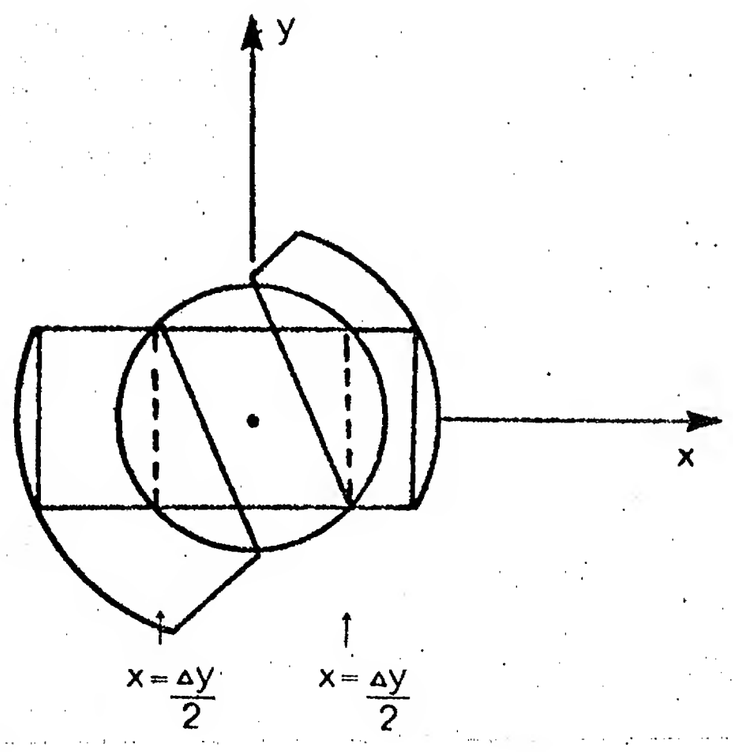


Figure 16. Computing a polyhedral approximation to the swept volume under pure rotation.

2. WEDGE — a polyhedral approximation to the volume swept out by a cuboid, aligned with the coordinate axes, as it rotates around the z axis, Figure 15. It assumes that the z axis does not penetrate the cuboid and that the rotation is less than π.

The input is a polyhedron, B, an axis of rotation which is the z axis of a reference frame and θ, the angle of rotation. The first step is to rotate the frame around z so that the x axis goes through the centroid of the projection of B on the (x, y)-plane of the frame. Compute an aligned bounding rectangular solid for B, $RB(B)$, whose dimensions are $(\Delta x, \Delta y, \Delta z)$. If the z axis does not pass through the object, then if $\theta < \theta_{\max} \leq \pi$ then simply return a WEDGE enclosing the swept volume. If the z axis penetrates $RB(B)$, then if $\Delta x > \Delta y$, cut B using the planes $x = \frac{\Delta y}{2}$ and $x = \frac{-\Delta y}{2}$, and return a cylinder of radius $\sqrt{2}\Delta y$ whose height is Δz and return the swept volumes of the pieces of B beyond the central area. The procedure is similar if $\Delta y > \Delta x$. Figure 16 illustrates this process. Here $\theta_{\max}$ is some user specified parameter, although it could be chosen to guarantee some kind of error bound. If $\theta > \theta_{\max}$, then divide the rotation into a set of successive rotations each returning a wedge.